Ontological Approach for Database Integration

Ph.D. Thesis

Nasser Alwan Alalwan

This thesis is submitted in partial fulfilment of the
requirement for the Doctor of Philosophy
Awarded by
Faculty of Technology
De Montfort University
United Kingdom, England
March 2011

Declaration

I declare that the work described in this thesis is original work undertaken by me for the degree of Doctor of Philosophy, at the Software Technology Research Laboratory (STRL), Faculty of Technology, at De Montfort University, United Kingdom. No part of the material described in this thesis has been submitted for the award of any other degree or qualification in this or any other university or college of advanced education.

Dedication

To my parents,

my wife,

my children Abdullah, Mohammed, and Munira,

my brothers and my sisters

The thesis is dedicated to my loving father, **Mr. Alwan Alalwan**, who has been a great source of motivation, inspiration and endless support throughout my life, and who sacrificed a lot for me to be what I am now.

It is also dedicated to my loving mother, who gave her love and support, for everything she sacrificed in her life for me. Without her loving care, prayers and support, it would have been very difficult for me to achieve my goals.

I owe everything I have achieved or will achieve to them.

I hope that by obtaining my PhD I can put smiles on their faces.

ACKNOWLEDGEMENTS

First of all, I would like to start by praising Allah (God) Almighty for all his bounties and blessings and for providing me with faith, patience and commitment to complete this research. Without him, none of this work would have been possible.

Also I would like to express my sincere gratitude to my supervisors, Professor H. **Zedan**, for his patient guidance and persistent support and encouragement throughout this research. Moreover, he gave me both the motivation for starting my new topic and freedom in my research interests.

I would like to thank Dr. F. **Siewe** for his expert guidance. I am grateful for his careful reading and constructive comments on our joint papers.

I would like to thank Professsor M. **Maskarinec** whose comments on my queries have added to the success of this project.

I would also like to express my thanks and appreciation to my friends, Dr. **Mohammad Taye** and Eng**. Sayed Saad-Aldeen** for their valuable comments, constructive criticism, scientific support, insightful comments, guidance and suggestions, without which this thesis could not have been produced in the present form.

I would also like to express my thanks and appreciation to my loyal friend Dr. **Ahmed Alzahrani** for his helpful advice.

I wish to thank all researchers, colleagues and staff of the STRL. During this work, I have collaborated with many colleagues for whom I have great regard, and I wish to extend my warmest thanks to all those who have helped me with my work in STRL. Especially I would like to thank my colleagues, Dr. **Mohamed Sarrab**, **Mohammed Al-Sammarrie, Abdulmalik Al-Hammad** and **Murad Megableh**, for their continuous encouragement.

I wish to express my love and gratitude to all my family, whose love and support have always been with me. In particular, I would like to give my special thanks to my beloved parents for their love and their continuing moral support throughout my studies. They had more faith in me than could ever be justified by logical argument. My special gratitude is due to my dearest brothers **Abdul-Aziz**, **Ibrahim, Abdullah, Weal,** and **Bader** and to my lovely sisters and their families for their loving support, concern and encouragement through all these years.

I would also like to express my thanks and appreciation to my father-in-law Mr **Mohammed Aleid** and my mother-in-law for their support, prayers, concern and encouragement through all these years. My special gratitude is due to my dearest brother-in-law **Khalid Alkhuraiji** for his help, concern and support.

Last, but certainly not least, I am indebted to my lovely wife **Gadah** for her endurance and unconditional support which provided vital encouragement during the period of my PhD study. Without her love and devotion, this research would have been impossible. Finally, I would like to mention my beloved children, **Abdullah**, **Mohammed**, and **Munira** who have given me happiness during the difficult period of my study.

Thank you all.

Leicester, England, March / 2011

Nasser Alalwan

ABSTRACT

Database integration is one of the research areas that have gained a lot of attention from researcher. It has the goal of representing the data from different database sources in one unified form.

To reach database integration we have to face two obstacles. The first one is the distribution of data, and the second is the heterogeneity. The Web ensures addressing the distribution problem, and for the case of heterogeneity there are many approaches that can be used to solve the database integration problem, such as data warehouse and federated databases. The problem in these two approaches is the lack of semantics. Therefore, our approach exploits the Semantic Web methodology. The hybrid ontology method can be facilitated in solving the database integration problem. In this method two elements are available; the source (database) and the domain ontology, however, the local ontology is missing. In fact, to ensure the success of this method the local ontologies should be produced. Our approach obtains the semantics from the logical model of database to generate local ontology. Then, the validation and the enhancement can be acquired from the semantics obtained from the conceptual model of the database.

Now, our approach can be applied in the generation phase and the validation-enrichment phase. In the generation phase in our approach, we utilise the reverse engineering techniques in order to catch the semantics hidden in the SQL language. Then, the approach reproduces the logical model of the database. Finally, our transformation system will be applied to generate an ontology.

In our transformation system, all the concepts of classes, relationships and axioms will be generated. Firstly, the process of class creation contains many rules participating together to produce classes. Our unique rules succeeded in solving problems such as fragmentation and hierarchy. Also, our rules eliminate the superfluous classes of multi-valued attribute relation as well as taking care of neglected cases such as: relationships with additional attributes. The final class creation rule is for generic relation cases. The rules of the relationship between concepts are generated with eliminating the relationships between integrated concepts. Finally, there are many rules that consider the relationship and the attributes constraints which should be transformed to axioms in the ontological model.

The formal rules of our approach are domain independent; also, it produces a generic ontology that is not restricted to a specific ontology language. The rules consider the gap between the database model and the ontological model. Therefore, some database constructs would not have an equivalent in the ontological model.

The second phase consists of the validation and the enrichment processes. The best way to validate the transformation result is to facilitate the semantics obtained from the conceptual model of the database. In the validation phase, the domain expert captures the missing or the superfluous concepts (classes or relationships). In the enrichment phase, the generalisation method can be applied to classes that share common attributes. Also, the concepts of complex or composite attributes can be represented as classes.

We implement the transformation system by a tool called SQL2OWL in order to show the correctness and the functionally of our approach.

The evaluation of our system showed the success of our proposed approach. The evaluation goes through many techniques. Firstly, a comparative study is held between the results

produced by our approach and the similar approaches. The second evaluation technique is the weighting score system which specify the criteria that affect the transformation system. The final evaluation technique is the score scheme. We consider the quality of the transformation system by applying the compliance measure in order to show the strength of our approach compared to the existing approaches. Finally the measures of success that our approach considered are the system scalability and the completeness.

PUBLICATIONS

1. Nasser Alalwan, H. Zedan, and F. Siewe, "Generating OWL Ontology for Database Integration", In proceedings of Third International Conference on Advance in Semantic Processing, 2009, Sliema, Malta, pp.22-31.

2. M. Taye, and N. Alalwan, "*Ontology Alignment Technique for Improving Semantic Integration*", In proceedings of The Fourth International Conference on Advances in Semantic Processing '*SEMAPRO 2010'* , Florence, Italy , October 2010.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DB	Database
DBMS	Database Management System
ER	Entity Relationship
EER	Extended Entity Relationship
RM	Relational Model
SQL	Structure Query Language
DDL	Data definition Language
DML	Data Manipulation Language
OWL	Web Ontology Language
AI	Artificial Intelligence
DAML	DARPA Agent Markup Language
DL	Description Logic
FOL	First Order Logic
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
OIL	Ontology Inference Layer
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
URI	Uniform Resource Identifier
URL	Uniform Resource Locater

UML	Unified Modelling Language
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language
SPARQL	Simple Protocol and RDF Query Language
SWRL	Semantic Web Rule Language
API	Application Programming Interface

CHAPTER ONE: INTRODUCTION

Objectives

- Enumerate the motivation for our research.
- Propose the general solution architecture.
- Provide the research methodology.
- Illustrate how the thesis is divided into chapters.

1.1 Introduction

This chapter provides an introduction to the thesis: beginning by describing the main problem addressed; followed by the research questions; a description of the overall research goals and objectives; an analysis of the main contributions of the research; and a description of the research methodology and practice. Finally, the structure of the thesis will be presented.

1.2 Problem Statement and Motivation

At the present time with the rapid growth in use of the Web, there is an increasing demand for the ability to effectively exchange data. Although current approaches have focused on exchanging data they have only partially succeeded due to the lack of semantics in the areas of data storing and data representation. Therefore a mechanism is required to enable the integration of data from many different sources whilst also retaining the capability of presenting them in a semantically uniform way.

Three big challenges must be overcome to realise this endeavour and upgrade the current Web. One of these stems from the Web representation language (HTML) and the second challenge originates from the nature of database which is responsible for data storing. The third challenge is the need to integrate data from multiple web sites.

To consider the first challenge; HTML is the language used to create Web pages and define its formatting specifications. One advantage in using HTML has been that it offers a simplified maintenance process, since it provides separation between data and the layout of the Web design, another advantage is the automated updating of Web content. However HTML legacy language suffers from the following problems:

- HTML is considered to be a machine-readable language; i.e. HTML concentrates on the information's appearance not on its meaning.
- Lack of semantic representations vocabularies.
- Parsing HTML is possible, however it is difficult to automate this since HTML forms could contain multiple frames (optional, multivalued, merged), attributes or attributes as values and (factored, duplicate) data.
- HTML is intended for user consumption; i.e. user "head knowledge" only. Consequently it is not considered as one of the semantic Web languages.
- Web sites change rapidly.
- HTML is a static language and the information dynamically generated at the time of user requests is stored in other places.

In the case of the second issue; database has many characteristics such as:

- Providing scalability.
- Facilitating powerful query language.
- Assisting rapid operations such as search and retrieval.
- Utilising the benefits of relational databases and management systems such as transaction management, security and integrity control.

Indeed, since vast amounts of Web information are stored in databases, they are therefore considered the backbone of the current Web. In addition, databases are responsible of producing the dynamic elements of Web sites. In fact, large portions of content referred to as "Deep Web" are stored in relational databases [66, 67, and 69]. More precisely about 77.3% of the data on the current Web is stored in relational databases [5]. However databases are not a knowledge representation language and

therefore they would not be a suitable replacement for HTML. It is also the case that database have certain problems with regard to the Web:

- Data stored in database is not accessible to Web search engines. The majority of the world's data today is still locked in data stores and is not published as an open Web resource (information on demand).
- Lack of semantic for naming tables and attributes.

The third challenge is the need to integrate the data that resides in relational databases in the Web environment, i.e. the Deep Web. This problem results from database heterogeneity, which includes many problems such as:

- Different specifications lead to different database designs.
- Designers have different viewpoints regarding concepts, structures and assumptions.
- Data itself may be stored in a variety of databases having different data models, formats and platforms.

The success of the Web crucially depends on semantic representation, and semantic database integration. Therefore, to address the first challenge current Web language needs to be upgraded from HTML legacy language to an expressive language, which is also a machine-understandable language. This need is considered to be one of main forces driving the move to invent a Semantic Web. A promising solution to the problem is to apply ontology languages, which are at the core of Semantic Web, since they satisfy the two conditions. An ontology language (e.g. OWL) can be used to annotate the current Web in order to add semantic to the data and prepare the information so that it can be processed. In the case of the second challenge using an ontology language to represent the semantic hidden in database structure is the best solution, since databases are not data representation language. To overcome the third challenge, database integration, using global ontology which plays the same role of domain schema in federated databases is the key solution. The methodologies typically used to seek to resolve database integration issues, such as a database warehouse and a federated database, [70] are inappropriate in this case since the need here is to create a system to

take care of both the semantic database and data representation in the Web too. Therefore the need is to produce a methodology that could integrate databases with the Semantic Web. For these challenges we propose using a hybrid ontology approach since it provides both semantic and shareability. Thus, it is appropriate to address the first two challenges using local ontologies and the third with a global ontology.

To ensure the success of the proposed approach the ontologies should be available. Domain ontologies are readily available whereas local ontologies are unavailable. The option of building local ontologies manually would be impossible due to the scale of the Web sites that might be expected to participate in the integrated system. Moreover the manual constrictions on creating new ontologies suffer are as follows:

- Labour-intensive.
- Error-prone.
- Time consuming.

Therefore we propose producing ontologies from relational database as an alternative technique to high cost manual ontology creation. The proposed approach benefits from reusing the vast amount of information available on databases and the semantic hidden in database schema. The goal in this case is to produce a methodology that can readily integrate databases and merge them with the Semantic Web. In order to accomplish the goal of interoperability ontology based architecture is sufficient, since ontology can explicitly and semantically describe data and this allows it to perform a significant role in resolving semantic heterogeneity problem.

The goal of this thesis is to integrate databases in web environment in order to enhance the current Web. This will include upgrading the current web through building ontologies from databases, since the isolated creation of ontology without consideration of databases will generate a need for a new method to create relationships between ontologies and databases.

It is significant that we seek to differentiate between data and database integration. While data integration involves joining data together that exists in different sources, providing users with a combined impression of this data, database integration involves

combining data that resides in relational and non-relational database sources in a unified manner. Therefore database integration is an aspect of data integration.

Now we can conclude reasons that motivate our research:

- Using the semantic Web for database integration requires the existence of ontologies. However, most ontology does not exist therefore we need an automatic system to construct ontology and to remove the excess burden of constructing ontology manually from scratch.
- Ontology design swings between subject description and size; ontology shapes its design in a direct and natural way, better than database model. However, the size of ontology is more often very large, since it has a large number of concepts which need numerous relationships to link them. Furthermore, these relationships are tangled and complicated. Consequently, the construction of a new ontology from scratch, without considering existing data storage (database), would not be a good candidate for a Semantic Web environment, because many Web sites rely heavily on storing their data in a relational database.
- Current database integration methods are concerned with syntax and structure schema heterogeneity, whereas the need is for a method to resolve the semantic heterogeneity problem, where ontology can be considered as a best solution [71].
- All existing ontology-based annotation tools can only be used to annotate static HTML pages thus far [47].

1.3 Research Question

The overall research question this thesis tries to answer is:

What is the best way to integrate databases utilising Semantic Web methodologies?

This question results in the following ones:

- **How can we bridge the gap between the database model and the ontology model?**

- **How can we integrate databases into a Semantic Web?**
- **How can we obtain high quality ontology from database sources?**
- **What is the best database source for producing better ontologies?**
- **How can we combine semantics obtained from more than one database source?**

1.4 Contribution to knowledge

The main contribution of this thesis is the development of a methodology that integrates both relational and conceptual models of databases in order to generate robustness ontology. The other major contributions of the study undertaken and reported on in this thesis can be briefly summarised as follows:

- Constructing a transformation system (relational model to ontological model).
- Creating a transformation system (conceptual database model to ontological model).
- Building a system to integrate both logical and conceptual database models to produce ontology.
- Implementing an automatic tool to transform SQL-DDL to OWL ontology.

Other aims and objectives (achievements) of this research are:

- Moving the representation of legacy system (database) to a richer semantic expression system (ontology) with preserving data.
- It is possible to reuse the existing database models to enrich the evolution of web ontologies, since both the Extend Entity Relationship (EER) model and relational schema are fully of an implicit domain knowledge which can be extracted in order to support the development of web ontologies.
- It is also possible to generate an ideal starting point for constricting complete, standard and effective ontologies that fulfil Semantic Web criteria.
- Discovering database semantics by employing reverse engineering methodologies.

- Identifying different database design, which includes the clarification of the differences between relational model and the extended entity relationship model.
- Identifying the differences between database modelling and ontology modelling so as to reduce the gap between the two.
- Combining the advantages from both the database schema and the database conceptual model which helps in identifying reliable categorisation patterns hidden in the data. Our fully automated algorithm is influenced through various types of semantic constraints filled with semantic.
- Developing a framework using the semantics of database schemas to develop a solution for bridging the gap between database and Semantic Web.
- A tool is developed to translate correct semantics obtained from a database model into an ontology model. A fully automated tool would not be able to capture semantics in both database modelling; however, we propose a solution which will require as little human interference as possible.
- Ensuring the transformation system from database to ontology can be applied to any application domain.
- Illustrating the effective use of our system through a case study.

1.5 Research Methodology

The research method used in this approach is a typical scientific research technique. As in the majority of the computer science approaches that describe research belonging to the constructive research field, the constructive approach refers to contributions to knowledge being developed as a new framework, theory, model or algorithm. The methodology of the proposed approach consists of four main steps and this section summarises the research methodology steps and outlines the reason behind using each part of them. The approach presented was realised by following the steps laid out below:

Step 1: critical engagement with existing literature

Background research was conducted initially with a theoretical literature review to enhance understanding of all the approaches related to the research question. To achieve

the required understanding from this stage papers in learned journals and digital resources were used.

Step 2: Architecture

This phase focused on the design of the framework architecture to capture the research objectives in order to solve the research question. This phase specified all components of the proposed framework and identified the technology used in the framework, explicitly stating how the framework components interact to achieve the research objectives.

Step 3: Algorithmic Development

This research step investigates each phase of ontology creation (classes, datatype properties, object properties, and instances). The main focus here was on providing a new algorithm that combines the advantages of both relational model and the conceptual database model to provide a flexible mechanism that will be applicable to any database source.

Step 4: Evaluation and conclusion

In this stage the prototype was built based on the proposed method and the experiments were conducted and the result evaluated. The evaluation step demonstrated the practical applicability of the research presented. A conclusion was provided from the experiences of the evaluation phase and a number of potential extensions for this research study were raised to motivate further investigation in the field of transforming database to ontology.

1.6 Thesis Outline (Organization of the Thesis)

The remainder of this thesis has been arranged according to the following framework.

Chapter 2: Database Overview

This chapter supplies an overview of database definition, structure, database modelling, advantages, and disadvantages.

Chapter 3: Semantic Web and Ontologies- State of the Art

Here we provide an overview of a Semantic Web, and introduce the definition and the structure of ontologies and ontology applied areas. Then the chapter explains ontology description languages such as Resource Description Framework **(**RDF), Resource Description Framework Schema **(**RDFS), and Web Ontology Language (OWL), and offers a clear comparison between them.

Chapter 4: Overview of Existing Approaches Transforming Database to Ontology

In this chapter the classifications are described based on sources from existing transformation systems. Then the current transformation approaches are examined, to capture the advantages and disadvantages of each approach.

Chapter 5: Approach Architecture

This chapter give an insight into general architecture systems. It also presents the main components of our framework with a full description, and describes how these components will interact with each other in order to provide good results. Following this, the disparities between database modelling and the ontological modelling are discussed in order to solve the transformation challenges and to bridge the gap between the two modelling systems. Then the existing sources are discussed, in particular to determine which of them will be able to produce a high quality outcome and why.

Chapter 6: Transformation from Database to Ontology

This explains in detail all the heuristic rules that are required for generating OWL ontology from a relational model written in Structure Query Language-Data definition Language (SQL-DDL) language. The rules are obtained by reverse engineering methodologies in order to generate a complete ontology structure.

Chapter 7: Translating an Extended Entity Relationship (EER) Model to OWL Ontology

In this chapter the proposed rules are able to translate an EER model to OWL Ontology. This chapter also provides a systematic method to utilise the ontology produced from the Relational Model (RM) source and augment it with further semantics obtained from the EER model.

Chapter 8: Case study and Prototype Implementation

This chapter offers a full description for our system prototype. In addition, it provides an experiment case study in order to reveal our system capability in producing high quality results.

Chapter 9: Evaluation

This chapter offers an expository comparison between our approach and the capabilities of other current approaches. The evaluation which takes place in this chapter compares the ontologies produced by different approaches with the ontology developed by domain experts as a benchmark test. Then we examine our approach against many success criteria, such as completeness, flexibility etc.

Chapter 10: Conclusion and Future Work

The final chapter concludes the work done in the thesis, presents the limitations of our work and offers suggestions for future work.

CHAPTER TWO: DATABASE OVERVIEW

Objectives

- Introduce the database: definition and advantages.
- Show data modelling in the database.
- Discuss EER model creation and criteria.
- Map EER model into relational model.
- Demonstrate SQL characteristics.
- Compare EER model to relational model.

2.1 Introduction

This chapter presents the main ideas relevant to the subject of databases. Firstly, it gives an overview of the database concept and related important features. Secondly, it classifies database types based on their data models. Thirdly, it focuses in detail on how to design relational databases. Fourthly, it presents a comparison between conceptual models and relational models in order to specify the best database representation source. Following this the final section concludes the chapter.

2.2 Preface

At the current time databases are indispensable, due to the many applications we use in this knowledge-based society. For example, search engines and websites utilise databases to store and identify information requests from people. Therefore, databases are now considered an integral part of the information revolution. The significance of databases is derived from decades of intensive research on the evolution of database design and technology [3].

A database can be defined as a collection of persistent data used by the application programs of some enterprise. Persistent data can be described as data that is truly stored in the database, as opposed to input or output data [1].

Each database is managed by a Database Management System (DBMS), shortly to become known as a Database System, which is a powerful tool for creating and managing large amounts of data efficiently. In addition to controlling data, the DBMS is also designed to deal with hardware, software and users. The functions of a database system include:

1- Dealing with user requests to:
    - Create new databases and specify the structure of the data.
    - Modify the data structure.
    - Query data.
2- Dealing with software to perform integrity and security checks etc.
3- Controlling hardware to physically store data.
4- Controlling data by showing how the data is stored, accessed, and linked to each other [3].

Generally, databases are useful due to the following characteristics:

1- Storing data once (centralised control).
2- Having multiple locations.
3- Making “backing up” easier.
4- Having multiple layers of security.
5- Being scalable.
6- Enforcing Standards.

2.3 Databases and data models

Each database must be built upon a data model, which is a "collection of concepts that can be used to describe data, data relationships, data semantics and consistency constraints" [2].

A data model for database design can be categorised into:

- Legacy models which were common in the past:
  - Network model.
  - Hierarchical model.
- Relational model which is the most commonly accepted model.
- New models:
  - Relational-object model.
  - Object model.

The classification of databases is based upon the data models. Therefore, each data model has its own database type, e.g. network database, relational database, object database etc.

In the last three decades the relational model, which is the heart of a relational database, has become the dominant model compared to old-fashioned models – the hierarchical model and the network model. This acceptance can even be applied to the new database models, such as object databases and object-relational databases. Now relational databases are popular and accepted by users, database designers and database vendors. Therefore, about 77.3% from data available today in the Internet is stored in relational databases [5]. Additionally the functions of and domain experts in relational databases are obtainable, whereas new models (i.e. object-relational databases or object databases) lack both these functions and domain experts.

2.4 Database Creation

The creation of relational databases has two phases. First, the conceptual model phase, which is done through either the Entity Relationship (ER) model or Extended Entity Relationship (EER) model. With the former model the designer specifies the entities, attributes and finally the relationships. The latter model includes all the steps identified in the former model, with the addition of some enhanced issues such as specialisation and generalisation etc. Second is the implementation phase, which initially converts the conceptual model to a relational schema, after which the implementation can be executed using Structure Query Language (SQL).

2.5 Phase-1 Conceptual modelling

A conceptual data model specifies the type of data the database stores, similarly to the way people perceive data in daily life. A conceptual data model uses concepts such as entities, attributes, and relationships. It hides the details of physical storage structures. This phase has three steps:

- Starting with an English description (requirements or specification).
- Developing a set of entities and relationships (ER model).
- Drawing an ER diagram (conceptual schema).

2.5.1 Entity Relationship model

The Entity Relationship(ER) model is a widely used data model for the conceptual description of databases. This model admits the specification of an enterprise, and then represents it with a high-level logical database structure. The ER model is significant for three reasons: first, it helps with mapping real world activities into a conceptual data model. Second, it is independent from database management systems. Third, it represents many semantic aspects of data meanings. Thus, the ER model can be considered to be one of the semantic data models [2, 3].

The Entity Relationship model can be represented by an ER diagram which is a diagrammatic notation associated with the ER model. It is used to express graphically

the overall logical structure of a database. Moreover, it focuses on schema representations rather than individual data. This criterion is helpful in designing databases, since the concepts are rarely changed whereas the contents are frequently changeable in entity types. Besides that an ER diagram can be characterised as a simple, clear, and widespread graphical model.

2.5.2 Basics of Entity Relationship model

An ER model includes three main elements:

- Entity;
- Attribute;
- Relationship.

2.5.3 Entity

An entity can be defined as a thing or object in the real world with self-sufficient existence; i.e. it is distinguishable from all other objects [4]. An entity might be a thing with natural existence (e.g. a person) or it might be a thing with a conceptual existence state (e.g. a job).

There are many terms that are usually used to describe an entity, such as entity type or entity set. We can define the entity type (entity set) as a data model representation that corresponds to a category of real-world objects. The entity is the actual set of values corresponding to the entity type, also called an individual entity.

Usually for simplicity the term ‘entity’ in the ER model is used to express the entity type, unless there is a need to distinguish between entity as individual or as type.

The entity type manifests in two forms [2]:

- **Strong entity**: this form does not need another entity to exist.
- **Weak entity**: the existence of this form depends upon another entity.

A classic example to demonstrate the difference between a strong and a weak entity is the entity 'child of an employee', the existence of which is based on the existence of the entity 'employee'. So we consider the employee a strong entity and the child of an employee a weak entity. Each entity has many properties that describe it and these properties can be represented by a set of attributes.

2.5.4 Attribute

An attribute is a descriptive property owned by each member of an entity set. Each attribute has a domain, and the domain of the attribute is a group of allowed values [4]. For example, the domain of the attribute x might be the group of all positive integers, or it might be a text of a certain length etc. So the attribute of an entity is a function that maps the entity into a domain. Also, in an ER model, each attribute has its own domain type. For example *Employee-Id*: integer, *Name*: string, and *Married*: Boolean.

The question that could be raised here is:

- How can we distinguish between the entity and the attribute?

The distinction mainly depends on two conditions:

- The structure of the real world enterprise being modelled;
- The semantics associated with the attribute [2].

For example, the 'Name' can sometimes be considered an entity for a telephone book database, whereas the *Name* of a student in a university database can be represented as an attribute.

2.5.4.1 Attribute classification

The attribute can be distinguished as follows:

- **Simple or composite attribute**:

  The attribute can be simple (atomic) which cannot be divided into sub-parts. However a composite attribute can be divided into sub-parts (many attributes)

which represent more basic (simple) attributes with independent meanings. The use of composite attributes depends on the user specification. For example, if the user wishes to refer to an entire attribute on some occasions and only a component of the attribute on other occasions. The benefit of using composite attributes is that it helps group together related attributes and thus makes the modelling cleaner. Additionally the composite attribute may appear as a hierarchy. For instance, the *Address* of a person is a composite attribute, although it could be considered an atomic attribute if its sub-parts do not have a valuable purpose, e.g. in designing a query. In this case, we can only refer to it as a unit. On the other hand, for a delivery company, the *Address* is considered a composite attribute because each part has a significant value.

- **Single-valued or multi-valued attribute**:

  The single-valued attribute would refer to only one value for a specific entity instance. However, the multi-valued attribute is an attribute which has a set of values for a specific entity instance. For example each person has only one name so the attribute *Name* will be a single-value attribute. Conversely each person may have more than one hobby. So the attribute *Hobby* can be considered a multi-valued attribute.

- **Derived attribute**:

  The value of this attribute can be deducted from the values of other associated attributes or entities. For example, the attribute *Age* is not necessarily stored. Still it can be computed if the required attribute *Date-of-Birth* exists.

- **Complex attribute**:

  A complex attribute is a composite and multi-valued attribute nested arbitrarily.

- **Null value**:

  A special value called Null is created when we do not know the value for an attribute, or the attribute is not applicable for the entity. For example, a *College-*

Degree attribute will be assigned the null value when applied to persons without college degrees, or when we do not know the person's specific college degree even if he has one.

- **Keys**:

  - ***Super key***:

    No two individuals (instances) in one entity are allowed to have exactly the same value for all attributes. Therefore each entity would have a unique identifier; in practice one or more of its attributes. Those attributes whose value uniquely identifies the entity will be the super key [5]. Thus, the super key attribute(s) distinguish the individuals from each other. However, the concept of the super key is not practical since it may contain extraneous attributes and this leads us to the idea of the candidate key.

  - ***Candidate key***:

    A candidate key is a super key for which no proper subset is a super key. The candidate key must satisfy two conditions:

    1. Uniqueness: no two tuples in the instances of an entity type have the same values for all the attributes of the key.
    2. Minimal: none of the attributes of the key can be removed without destroying the uniqueness property.

  - ***Primary key***:

    A primary key is a candidate key that is chosen by the database designer. Here the database designer considers the following criteria when choosing the primary key of an entity:

    - Unique and minimal.
    - Not null: it must have value.

- Protects sensitive data.
- Invariant: it is rarely or never changed.
- Meaningful key.

- ***Alternate Key:***

  An alternate key is a candidate key that has not been chosen to be the primary key.

2.5.4.2 The primary key of strong and weak entities:

Each entity must have a primary key to distinguish each instance from the other. A strong entity has sufficient attributes to form a primary key. In contrast, a weak entity does not have sufficient attributes to form a primary key, except for a discriminator, which is a set of attributes that allows a distinction to be made in the case of one particular strong entity. In other words the discriminator uniquely identifies one single individual of the weak entity for one individual of a strong entity - thus the discriminator could only be considered to be a partial key. Therefore the weak entity must be associated with another entity called the identifying (owner) entity to be meaningful. To construct the primary key for the weak entity, we combine the primary key of the identifying entity with the discriminator of the weak entity.

2.5.5 Relationship

A relationship is the association of several entities [1]. Each relationship is defined according to degree, which shows the number of entities that participate in that relationship. Therefore the relationship can be classified based on relationship degree as below:

- Unary relationship: this is a relationship between an entity and itself.
- Binary relationship: this is the most often used relationship, between two entities.

- N-ary relationship: relationships exist between N entities, where N>2. The most popular case of N-ary is the ternary relationship, which is when a relationship exists between three entities.

Two different constraints can be applied to a relationship: cardinality ratios and participation constraints.

2.5.5.1 Cardinality ratio constraints

Cardinality ratios could be defined as the number of individual relationships that exist between the participating entities [2]. In other words the cardinality ratio constraints declare the number of individuals to which another entity can be linked via a relationship. Cardinality will take one of the following forms in mapping between entities:

- One-to-One.
- One-to-Many.
- Many-to-One.
- Many-to-Many.

The appropriate mapping cardinality for a particular relationship obviously stems from the real world situation.

Besides the relationship mapping types above, there are some other types of relationships such as:

- Functional relationships: relationships such as “AVERAGE” or “SUM” may exist between entities. However these are not possible to represent in the relational model.
- Recursive relationships: relationships such as “ANCESTOR” may be recursive. This is not possible to implement directly in a relational model.

2.5.5.2 Participation constraints

The second constraint is the participation constraint which specifies whether the existence of an instance of an entity type depends on another instance existence via a particular relationship or not. This constraint defines for each entity type the minimum number of participant relationship instances. The participation constraints can come in two forms:

- ***Total***: Every individual entity must participate in the relationship. For example each relationship between the owner entity and the weak entity (i.e. identifying relationship) always has total participation constraints towards the weak entity.
- ***Partial***: Some individual entities can participate in the relationship.

Another way to express the participation constraints is by using the **(min, max)** notation of Jean-Raymond Abrial. This notation involves associating a pair of integer numbers for specifying structural constraints on relationships which are used to specify the minimum and maximum participation of each entity. When the minimum is 0 the implication is that the relationship will have partial participation, whereas if the minimum is > 0 this implies total participation [4].

2.6 Extended Entity Relationship (EER)

Many database schemas for database applications can be adequately represented by the ER model. However, the higher concepts are not covered by the ER model which needs to be designed to express some advance constraints and data properties in a proper, accurate and precise way in order to reflect newer applications requirements in database technology. Therefore there is a need for new concepts such as inheritance, specialisation, and generalisation. These concepts will be integrated into the ER model to form the new conceptual data model. For this purpose the Extended Entity Relationship, also known as Enhanced Entity Relationship (EER), was developed for adding semantic concepts to the ER data model.

2.6.1 Specialisation

Specialisation is the process of designing sub-groupings within entity types. An entity type includes sub-groupings of entities that are distinct in some way from other entities in the set. In other words, a subset of entities within an entity type may have attributes that are not shared by all the individuals in the entity type [2, 4]. Hence specialisation can be applied when there are sub-groupings of individuals for an entity type, where each group shares the same meaning and can be distinguishable from other groups. Therefore, if sub-groupings are of significance to the database application, then it can be explicitly represented by specialisation; otherwise there is no need to complicate database design.

An entity type can be categorised by using one or more distinguishing features. When more than one specialisation is formed in an entity type, a particular entity may belong to multiple specialisations. In term of the EER diagram an ISA relationship is used to depict the specialisation. Moreover the ISA relationship may be referred to as a superclass-subclass relationship. The entity type sub-groupings are called subclasses, and the entity type itself is called the superclass.

- **Why do we need to add specialisation to the ER model?**

There are two reasons for adding specialisation to a data model, i.e. superclass-subclass relationships: The first reason is that the superclass might have certain attributes shared by only some instances. The second is that members of subclass entities participate in a particular relationship type where the other entities do not [4].

2.6.2 Generalisation

Specialisation represents a top-down design process. In the bottom-up design process, multiple entity types are synthesised into a higher-level entity type on the basis of common features. If two or more entity types have several attributes with the same concepts across them, this can be expressed by generalisation, which is a containment relationship that exists between higher level entity types and one or more lower level entity types. Higher and lower entity types also may be designated by the terms

superclass and subclass respectively [2]. Generalisation could be considered a process of abstraction. So the steps in generalisation are:

- Suppress the differences between several entity types.
- Identify their common feature.
- Generalise them into one single superclass.
- Make the original entity types subclasses.

For all practical purposes generalisation is a simple inversion of a specialisation. In terms of the EER diagram itself we do not distinguish between specialisation and generalisation.

- **Attribute and relationship inheritance**:

Attribute inheritance is a significant property of higher and lower level entities. It is created by specialisation or generalisation. That is to say, the attributes of higher entity types are inherited by lower level entity types. The attribute inheritance applies through all tiers of lower level entity types. Additionally, lower level entity types inherit participation in the relationship types in which their higher level entity participates. In a hierarchy of entity types, lower entity types may be involved in only one ISA relationship; that is entity types have only a single inheritance. If an entity type is a lower level entity type which has more than one ISA relationship, then the entity type has multiple inheritances and produces a resulting lattice structure [2].

2.6.3 Constraints on generalisations and specialisation

We discuss constraints that apply to a single specialisation or a single generalisation. In brief, we refer only to generalisation even though it applies to both of them.

2.6.3.1 Membership

This constraint determines which entities can be members of a given lower level type. There are two kinds of membership:

i. **Condition-defined**: The membership of lower level entity types is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate. In other words the entity will become a member of a subclass by placing a condition on the value of some attribute of the superclass. If all lower level entities are evaluated on the basis of the same attribute, the generalisation here is known as 'attribute-defined'.
ii. **User defined**: In user-defined situations, lower level entity types are not constrained by a membership condition; rather the database user assigns entities to a given entity type [2, 4].

2.6.3.2 Disjointness constraints

This determines whether or not an entity may is a member of more than one lower entity type within a single generalisation [2, 4]. Disjointness can occur in two situations:

i. **Disjoint**: This constraint requires that an entity belongs to no more than one lower level entity type.
ii. **Overlapping**: The same entity may belong to more than one lower level entity type within a single generalisation. The overlapping constraint is the default.

2.6.3.3 Completeness constraints

This specifies whether or not an entity of the higher level type must belong to at least one of the lower level entity types within a generalisation [2, 4].

i. **Total generalisation**: Each higher level entity must belong to a lower level entity type.
ii. **Partial generalisation**: Some higher level entities may not belong to any lower level entity type. This is the default.

The disjointness and the completeness are independent so there are four possible constraints on generalisation:

i. Disjoint and total.
ii. Disjoint and partial.

iii. Overlapping and total.

iv. Overlapping and partial.

2.7 Phase-2 Implementation Phase relational model-structure query language (SQL)

The implementation phase can be carried out using the Relational Model (RM). The relational model uses a “collection of tables to represent both data and the relationships amongst these data, also called the relational schema” [1]. Subsequently, the relational schema can be implemented in the Structure Query Language (SQL) [3].

2.7.1 Structure of Relational databases

Here we present the relational model elements with their equivalent SQL terms:

- **Relation (table)**

  The relational data model takes its name from the mathematical concept of 'relation' which corresponds to the concept of 'table'. A relational database consists of a collection of tables each of which is allocated a unique name.

- **Tuple (row)**

  A row in a table represents a relationship among a set of values. The order in which tuples appear in a relation is irrelevant.

- **Attribute (Column)**

  The column header is the attribute; for each attribute, there is set of allowed values called a domain.

- **Keys**

  - Primary key: a super key with minimal attributes.

- Foreign key: a primary key of another relation which represents the relationship between relations.

- **Database schema**

  The database schema is the logical design of the database, as distinct from a database instance which is a snapshot of the data in the database [3].

2.7.2 The algorithm of mapping the ER model to the relational model

This mapping algorithm converts the basic ER model into the relational model over many steps. These steps consist of mapping entity types (strong and weak), binary relationships, N-ary relationships and attributes (simple, composite, and multi-valued) etc. [2].

2.7.2.1 Mapping the entities

i. Mapping of strong entity types

Each strong entity type creates a relation that includes:

- All simple attributes of the strong entity.
- Simple component of composite attributes of the strong entity.
- The primary key chosen in the ER will be the primary key of the relation. The candidate keys can be considered to be unique or index keys.

ii. Mapping of weak entity types

For each weak entity type, create a relation and include:

- All simple attributes of a weak entity.
- Simple component of composite attributes of the weak entity.
- The primary key of the relation corresponding to the weak entity will contain the primary key of the owner entity type, in addition to the partial key (discriminator) of the weak entity type.

2.7.2.2 Mapping the relationship

Here we use the term relation which represents the participating entity type in all the following mapping procedures.

i. Mapping of binary 1:1 relationship types

There are two possible approaches:

- ***Foreign key approach***

  This is the most used approach. First we add the primary key of the partial participation relation to the relation with total participation. Then we include all simple attributes and simple components of composite attributes of the relationship type as attributes to the relation with total participation.

- ***Merged relation approach***

  When both relations participate totally in the relationship it is possible to combine the two entity types and the relationship into a single relation.

ii. Mapping of binary 1: M relationship types

This is for each binary relationship (cardinality of 1: M) that does not involve weak entities. We identify the relation at the M-side of the relationship type; include as the foreign key to this relation the primary key of the relation at the 1-side participating in this relationship.

iii. Mapping of binary N: M relationship types

For each binary N: M relationship type we create a new relation that includes:

- Simple attributes or simple component of the composite attributes of the N: M relationship.
- The primary keys of the relations participate in the relationship as foreign keys.

- The combination of the foreign keys will form the primary key of the new relation.

iv. Mapping of N-ary relationship types

For N-ary relationship types, where N>2, create a new relation to represent this relationship which includes:

- Any simple attributes or simple component of the composite attributes of the relationship.
- The primary keys of the relations participating in the relationship as foreign keys.
- The primary key of the new relation will be the combination of all the foreign keys which refer to the relations that participate in this relationship.

v. Mapping unary relationship

When a relationship exists between the entity type and itself, then we add the primary key of the relation to the relation itself as a foreign key with a new name.

- Mapping the multi-valued attributes

For each multi-valued attribute, create a new relation. This relation will include:

- The multi-valued attribute.
- The primary key of the owner relation as a foreign key plus the multi-valued attribute.
- The new relation (multi-valued relation) primary key is the combination of both the multi-valued attribute and the primary key of the owner relation.

2.7.3 Structure Query Language

One of the major reasons for the success of relational databases is the use of SQL language, because SQL is a standard for all relational database systems. Thus, a

database designer is less concerned when converting from one database system to another since both systems follow the same language standards. In practice, there are many differences between commercial database systems, however most systems truly support the standard SQL [1, 3].

SQL can be categorised into two types, Data Definition Language (DDL) and Data Manipulation Language (DML). SQL-DDL includes creating schemas, tables and data types. However, SQL-DML is used for managing data within schema.

SQL has many features. For example it is based on relational calculus and it includes some features from relational algebra, although SQL syntax is more user-friendly than both of them. SQL is also a comprehensive database language, since it includes DDL, DML languages and queries. Furthermore SQL provides a higher declarative language interface which makes the user less concerned about the actual execution.

2.7.3.1 SQL commands related to our work

- *CREATE TABLE*: is used to specify
  - Table name to represent a relation.
  - Attributes: name, data type, domain values, and attribute constraints.
  - Initial table constraints: keys, referential integrity.
- *INSERT INTO***:** is used for inserting records into tables.

2.7.3.2 Attribute data types in SQL

- Numeric data types include:
  - Integer numbers of various sizes (INTEGER or INT and SMALLINT).
  - Real numbers of varying precision (FLOAT, REAL and DOUBLE).
- Character string:

- Fixed length (CHAR, or CHARACTER).
- Varying length (VARCHAR).

- Boolean data type has values of TRUE or FALSE.
- DATE and TIME and TIMESTAMP.

2.7.3.3 Attribute constraints

- NOT NULL: if NULL value is not permitted for the attribute.
- DEFAULT: a default attribute value for a new tuple.
- CHECK: to restrict attribute values to clause conditions.

2.7.3.4 Keys

- PRIMARY KEY
- FOREIGN KEY
- UNIQUE

2.7.3.5 Referential integrity

- ON DELETE CASCADE
- ON UPDATE CASCADE

2.8 Comparison between Entity Relationship and Relational Model

Both the extended entity relationship model and the relational model are abstract logical representations of real world enterprises. Since the two models employ similar design principles, we can convert an EER model into an RM. However not all the semantics present in the EER model can be translated to the RM. For example some information might be lost, such as inverse role relation and composite attributes while translating ER

diagrams into a relational schema [6]. Another issue is that the information might be represented implicitly, which means that one design can represent more than one case such as class hierarchy, multiple inheritance etc.

Moreover, the ER model and the RM model are different in their representation of a connection between entities. The ER model uses a relationship to express explicitly that connection, while the RM model represents the connection via an attribute in an implicit way by putting the primary key in a relation as an attribute of another relation. This implicit way of representing a connection results in the loss of more semantics in the process of transforming from the ER model to the RM model.

In addition, the EER model is different from the RM model since the EER model shows explicitly some semantics such as specialisation and generalisation. In contrast, the RM model shows specialisation implicitly or indirectly, that is, all individual entities will be stored in one entity table. Query statements are then used in SQL to determine the membership of each individual entity, so only the query conditions can categorise each individual entity. However, the case of generalisation is not applicable because each attribute that appears in different tables will have a different label name. The semantics of commonality are impossible to infer [2].

2.9 Choosing approach source between database models

This section explains the strength and weakness criteria in different database modelling in order to choose the suitable sources for our approach. This comparison will help us to understand the abilities of each model and how can we relate database models to ontological concepts. In the comparison of all database models, the conceptual models can be represented by the ER diagram or the EER model, and the relational model can be represented by the relational schema or the SQL-DDL language. Table 2.1 shows a comprehensive comparison between the four models.

Table 2-1: comparison between different database modelling's languages

Model	Conceptual Model		Relational Model	
Criteria	ER	EER	RM Schema	SQL-DDL

Model		Conceptual Model		Relational Model	
General modelling concept		Conceptual model is closer to the "semantic" ontology design.	Conceptual model is closer to the "semantic" ontology design.	Strong mathematical foundation closer to the description logic (the basis of OWL ontology).	Is built upon RM model and is more powerful, since it adds some expressive power that is not available in the theoretical RM.
Availability		Hard to obtain	Hard to obtain	Can be obtained indirectly	Easy to obtain
Design		Graphical representation is difficult to parse. (It can be converted into a set of formal definitions).	Graphical representation is difficult to parse. (It can be converted into a set of formal definitions).	Facts easy to parse	Definitions easy to parse
Representation of Concepts (entity, relationship)		Clear distinction between entity and relationship	Clear distinction between entity and relationship	Tables mixing up entity and relationships	Tables mixing up entity and relationships
Participation and Cardinality		Explicit declarations for participation and cardinality of the relationships.	Explicit declarations for participation and cardinality of the relationships.	NA	Foreign key can represent cardinality in an implicit way by uniqueness and not null; however, the exact cardinality restrictions cannot be obtained from the SQL/DDL-code.
Inheritance		NA	The best modeling for inheritance concept	Indirect representation through foreign keys	Indirect representation through foreign keys
Instances		Not available	Not available	Not available	Available
Attribute characteristic	**Domain of attribute**	Not available	Not available	Available	Available and attribute domains can be more specific than RM schema
	Keys	- Keys (super key, alternative key, primary key, discriminator(partial key)	- Keys (super key, alternative key, primary key, discriminator(partial key)	Primary key, discriminator, foreign key, however alternative key indirect way	Primary key, discriminator, foreign key, however alternative key indirect way
	Attribute constraints	NA	NA	NA	Explicit declarations (using more complex techniques in defining constraints, such as the use of assertions or triggers)

Model		Conceptual Model		Relational Model	
	Enumerated attribute	NA	NA	NA	Possible
	Attribute value restriction	NA	NA	NA	Possible
	Sub property	Possible	Possible	Not possible	Not possible
General disadvantages		A lot of changes in the implementation level do not update in the original ER.	A lot of changes in the implementation level do not update in the original EER.	-Tables do not represent entity but also relationship, multi-valued attribute. -Not clear hierarchy (fragment tables look like ISA relationship)	- Using the theoretical design of RM. So it has some of the disadvantages that appear in RM.

The conceptual model in database can be divided into two models ER and EER, where the logical model was represented by a relational schema in the past and SQL-DDL depicts the physical model now. From Table 2.1, it is obvious that both the conceptual model and the logical model share some of the concepts with the ontological model. It is easier to obtain semantics from logical model since each database stores the source code of its table's definitions. However the conceptual model is represented by a graphical model whereas the relational model is represented by facts or definitions. One of the criteria that distinguish between the two models is the clear distinction between concepts and relationships in the conceptual model; the logical model uses tables to represent concepts and some relationships' cases. Also, only the EER model from the four database models has a clear inheritance hierarchy. The cardinality constraints are rich in the conceptual model, in contrast with the relational model which represents the cardinality in an implicit way. The relational model represented by SQL-DDL is the best way to describe attributes' characteristics and preserve the data.

After balancing the four models we reach the conclusion that our system needs a source that is always reliable. Reliability here includes two conditions; the first is the accessibility of the source, and the second is its ability to represent the current database. These two conditions are available only in the SQL source, and, in addition to all the

other SQL features it is easy to parse, which supports automating our approach. However in order to reach all the semantics of the database we utilise the EER source to validate and enhance the ontology produced by SQL.

2.10 Reasons for choosing the relational database instead of using the object database

There are many disadvantages for using object databases, as listed below:

i. Any changes to the schema caused by creating, updating or modifying a persistent class will necessitate a system recompilation.
ii. It can take a long time to update all the instance objects depending on the size of the database.
iii. Language dependence - data only accessible from a specific language using a specific Application Programming Interface (API).
iv. Lack of ad hoc queries i.e. the query in the object database is dependent on the system design, so the query is customised by the designer, which leads to reduced flexibility in the object database. In other words ad hoc queries go against the principle of encapsulation.

2.11 Summary

In this chapter we introduce the database definition and the advantages of using it. After that the focus on the data modelling effect in database modelling is discussed. Then the chapter concentrates on the conceptual model of database by explaining the design steps of both ER model and EER model. Then the mapping algorithm from conceptual model to relational model was elaborated. Then the comparisons between database modelling were held to show the strengths and the weaknesses in each model. Finally the chapter provides the reasons behind choosing the logical model written in SQL-DDL script over other database models.

CHAPTER THREE: SEMANTIC WEB AND ONTOLOGIES- STATE OF THE ART

Objectives

- Define semantic web and ontology.
- Introduce ontology structure and languages.
- Show ontology applications.

3.1 Introduction

This chapter gives a brief introduction to the subject of the Semantic Web, as well as definitions of ontology. Also in this chapter, many ontology languages are explained in order to give an indication of the strengths and weaknesses of each language, which will help in determining the most suitable language for our purpose. The main concepts and structure of ontology are provided in order to specify the criteria of ontology design. Finally, the usability of ontologies in different software areas is discussed.

3.2 Semantic Web

The easiest definition of Semantic Web is a Web with meaning or, to simplify, it can be seen as a Web of data where the data can be shared and cannot be owned by a standalone application. Most people believe that the Semantic Web is merely a way of making connections between web pages. However, the Semantic Web is concerned with establishing relationships between different data objects in a Web distributable environment. Moreover, Semantic Web determines data properties.

The Semantic Web is an extension of the current World Wide Web. Its main purpose is to allow people to share information, regardless of which application or website they are using. In fact, the Semantic Web will provide a common framework which makes data available for reusing and sharing. This can be done applying two methods. The first

method is to create common formats for data from different sources. This method is different from the way current websites exchange information because data in Semantic Web environments is ready to be integrated. The second method involves generating languages that show the relationship between data and real world objects. Current efforts are being done by researchers and other businesses (e.g. W3C) to produce such a general framework and languages for the Semantic Web with the ability to share and reuse data. Tim Berners-Lee (WWW inventor) stated that the Semantic Web is "an extension of the current Web, in which information is given a well-defined meaning, better enabling computers and people to work in cooperation." [8].

Therefore, the Semantic Web [16, 28] has the ability to represent data in a rich, meaningful and readable form, which can be utilised by both humans and machines. Furthermore, in order to facilitate knowledge sharing, the Semantic Web provides different services such as publishing, discovering, and automatic annotation. Consequently, many tools and applications have been implemented to achieve the interoperability aim of Semantic Web.

3.2.1 Semantic Web layers

The Semantic Web [31] has evolved to cope with the heterogeneous and the distributed environment of the Web. To ensure the success of upgrading the current Web to a superior standard, the content should be reformatted in a machine understandable style. Data annotation is the mechanism which facilitates this aim. The Semantic Web architecture is depicted in Figure 3.1.

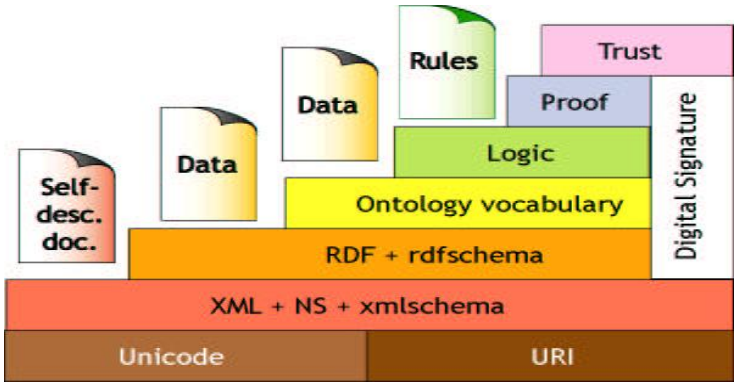


Figure 3-1: Semantic Web Architecture [8]

The description of architecture layers is clarified below [8, 16, 28, and 31]:

- **URI and Unicode**: these layers have two functions: first, identifying objects and resources available in the Web, and locating them; a uniform system of identifiers (URIs) can facilitate these functions. They are also responsible for granting unique names to the resources. The Unicode is a standard encoding system which represents computer characters.
- **Extensible Markup Language** (XML) is a machine-readable markup language. One of the many reasons for this language being widely accepted in the WWW community is the flexibility of its format. Therefore it facilitates the current Web services such as e-businesses and e-commerce. It also helps in expanding the activity of software. XML Namespace is the document declaration. Users utilise the freedom of the XML format in arranging their documents with any arbitrary structure. However, the XML structure does not provide any semantic indication. In fact, XML language can be utilised by its syntax only.
- **Resource Description Framework** (RDF) is considered to be the first Semantic Web layer. Since RDF can describe some semantic representation for information, metadata about Web resources will gain machine- accessibility. In fact, RDF is a graphical model that can utilise the URI Web resources definitions in representing relations among resources. The developed version of RDF is an RDF Schema which supplies the class hierarchy structure and the relationship between classes and objects. A deficiency in the RDF Schema stems from relying on informal semantic representation for some RDF Schema primitives which eventually cause error-prone interpretation.
- **Ontology Vocabulary** is a language layer which describes data grammatically by use of common vocabularies. This layer is concerned with both syntax and semantic representation of data in order to create ontologies that allow the inference process. In short, ontology satisfies the aim of providing data with a semantic description in a uniform method, thus making the exchange of information easy between different parties. There are many languages currently proposed to represent ontology. DARPA Agent Markup Language and Ontology Inference Layer **(**DAML+OIL) language is an extending to RDF and RDF

Schema. Furthermore DAML+OIL language has rich modelling and efficient reasoning since it is built on description logic. Web Ontology Language (OWL) is another example of an ontology language which has inherited all the features of DAML+OIL language and whose logic is developed from Description Logic. More details about ontology languages are presented in section (3.3.8)

- **Logic and Proof**: these are Semantic Web layers in which the structure of ontology is considered while building the systems. To ensure consistency and remove any redundancy that might occur in some ontology concepts, a reasoner can be used for checking and resolving these purposes. Also, the reasoner can be utilised in producing new inference results. The difference between the logic layer and the proof layer is that the former is involved in producing rules for inferences, whereas the latter is more concerned with providing clarifications for the result deduced by a reasoner. A reasoner is usually used as a built-in function in Web agents which need to translate its techniques into a proof representation language.
- **Trust** is the top layer of the proposed Semantic Web architecture. This layer concerns the reliability of Web information. Consequently, the quality will be assured.

3.3 Ontology

Today the Internet websites need to be enhanced by formal semantic representations for the current web content. When this happens, the Web will be semantic and suitable for both human and machine use.

One of the Web's roles is to build a source of reference for information on several subjects, while the Semantic Web is intended to annotate the current Web with semantic meaning. The essential part that facilitates the Semantic Web is ontology, since it allows distributed systems to share information. This means that ontologies can help unrelated applications to collaborate with each other.

3.3.1 Ontology Definition

Gruber defined ontology early on as "an explicit specification of a conceptualisation" [48]. Since then the definition has been amended to express other features. The new definition of ontology is: a formal explicit specification of a shared conceptualisation [26], where:

I. **Formality** means building ontologies in machine understandable language; it should be defined using logic-based languages. Formality is essential to eliminate the vagueness of informal notations (e.g. natural language).
II. **Explicit specification** stands for providing descriptive names (i.e. terms) for ontology concepts and clear characterisation for the constraints on these concepts. It also includes the definition of the relation which shows how a concept relates to other concepts.
III. **Shared** indicates that different communities across the Web are generally agreed on the meaning of domain concepts (consensual knowledge), thus allowing applications to exploit and reuse ontologies.
IV. **Conceptualisation** means ontology concepts appear in comprehensible form; i.e. domain knowledge is defined in an abstract model. This abstract model focuses on how to capture people's ideas about objects from real world, usually in a specific subject in a particular domain.

3.3.2 Ontology Objectives

- To agree on the understanding of the shared concepts of information structure among different communities.
- To make a domain knowledge analysable and reusable.
- To specify domain assumptions in an explicit manner.
- To make a partition between operational knowledge and domain knowledge.
- To analyse domain knowledge.

Those objectives [107] can only be applied on domain ontologies (i.e. global ontology). While application ontologies (i.e. local ontologies) can benefit from these characteristic after mapping them to the global ontologies.

3.3.3 Ontology Representation

Ontology consists of four principal elements: concepts, relations, axioms, and instances. The definitions of each element are presented below:

- **A Concept** (also known as a term or a class) is the essential abstract component of a domain. Typically, the class represents a group of common properties owned by many members. Also, classes are arranged in hierarchical graphs on two levels. Higher level classes are called parent classes and the subordinate levels are called child classes. A graph of concepts might organise classes in a lattice or taxonomic view; for example, the class 'Faculty' could have many subclasses, such as 'Department' and 'College'. Moreover, the concepts might have many different distinguishable properties.
- **A Relation** (also known as a slot) is used in ontology structure to provide a declaration for the relationships between concepts in a specific domain. In order to specify the two classes involved in a particular relationship, one of them will be described as a 'Domain' and the other one as a 'Range'; for instance the relationship 'Work' can have the concept of 'Employee' as a domain and 'Faculty' as a range.
- **An Axiom** (sometimes called a facet or role restriction) is utilised in ontology by forcing restrictions on values of both classes and instances. Logic-based languages, such as first-order logic, have been developed in order to express these constraints. Furthermore these languages can be facilitated as the verification process for the consistency of ontology structure.
- **An Instance** (also known as an individual) is a relationship between ontology concepts and relation and their real values; for instance, 'Saudi Arabia' could be an instance of the class 'Asian countries', or simply 'countries'.

3.3.4 Structure of Ontology

The general and formal structure of ontology is demonstrated as follows:

$$5\text{-}tuple\ O\text{:} = (C, H^C, R, H^R, I),$$

Where:

- C: refers to a set of concepts ("*rdf:Class*"). These concepts are organised in a class/subclass (subsumption) hierarchy.
- R: stands for a set of relations that connects concepts ("*rdf:Property*"). All relations in ontology are binary types between only two concepts.

$$R_i \in R \text{ and } R_i \rightarrow C \times C.$$

- H^C: represents the hierarchy of ontology concepts by using the special case of relation. ("rdfs:subClassOf").

$$H^C \subseteq C \times C,$$

where $H^C (C_1, C_2)$ denotes that C_1 is a sub-concept of C_2.

- H^R: depicts a relation hierarchy in the form of a relation ("*rdfs:subPropertyOf*").

$$H^R \subseteq R \times R,$$

where $H^R (R_1, R2)$ denotes that R_1 is a sub-relation of R_2

- I: supplies the concepts with instances in a particular domain ("rdf:type").

Generally, taxonomies, thesauri, and ontologies are types of methods that represent the Semantic Web classification of domain concepts.

First, taxonomy is a way of organising the vocabulary of concepts in tree form or hierarchical models; this includes an explicit definition of domain concepts and their relationships. In the taxonomy method, only the relation of "subclass" and its inverse

"superclass" are allowed. A real world example which can be considered as taxonomy is Yahoo! categories.

Second, thesaurus is used to represent vocabularies while considering relations between terms [25]. In fact, these terms are the relationships obtained by taxonomy methods with richer conceptual descriptions. Therefore, a thesaurus can be considered as an extension to the taxonomy method with more semantics and expressive features. Therefore a thesaurus includes some relationship characteristics, such as homography, equivalence and hierarchy. Any lexicon database, such as WordNet, can be considered as a thesaurus, since it captures concepts and preserve their semantic relationships.

Third, ontologies extend the taxonomies idea with rich relations between concepts and properties; also the axioms play a significant role in presenting ontology restrictions. Moreover, ontologies describe a domain by defining its concept set; therefore they can be seen as the skeletal establishment for a knowledge base.

The ontology community categorises ontologies in terms of their structure into two types: lightweight and heavyweight ontologies. Both types share some contents which include concepts, relationships between concepts, and attributes. However, axioms and constraints are considered only in heavyweight ontologies. Many kinds of languages can participate in implementing both lightweight and heavyweight ontologies [18]. In terms of language formality, ontologies can be divided into four groups:

- *Highly informal:* mostly described by natural language; however, this type of language might not be considered as an ontology if it lacks machine-readability;
- *Semi-informal:* a natural language used to describe ontology structure, and it can be utilised as machine-readable language;
- *Semi-formal:* a language such as RDF expresses ontology structure and some restrictions in formal definitions;
- *Rigorously formal:* ontologies provide definitions of terms with formal semantics, as well as properties with essential characteristics; also, it shows a notion of a completeness theorem (e.g. Web Ontology Language OWL) [106].

Also ontology can be classified to global and local ontology. Global ontology can be seen as shared vocabularies between many local ontologies. Also, it represents the global schema of a domain. However, local ontologies concentrate on different sub-domain fields. Moreover, a local ontology function is to express more specific information whereas a global ontology contains only general level of concepts. Finally, a local ontology allows different groups to use their own terminology whereas a global ontology utilises the agreed terminologies.

In fact the expressiveness of ontology measured by the degree of explicit metadata is captured from the domain knowledge. Usually, ontologies try to catch the semantics of a particular domain, and the more relations and constraints can be captured, the more ontology may be considered expressive. The important factor that affects the expressiveness of ontology is language specifications which limit the language abilities [32]. In order to provide ontology with a formal semantic structure, it should be expressed in knowledge-based language. After that, ontology can capture domain knowledge specifications and become machine-processable, and eventually appear in a well-defined design [8].

3.3.5 Criteria for ontology design

We need objective criteria to control the ontology design, since each designer makes his own design decisions in representing anything in the ontology. Therefore the criteria given below can be utilised in assessing ontology design.

- **Clarity**:

  The ontology should express the intended meaning of the defined terms. In order to reach this aim, the definitions should be independent from social cases or computational specifications.

- **Coherence:**

  Coherence implies that the inferences should be consistent with the ontology definitions.

- **Extendibility:**

  New concepts can be defined while considering the existing terms without requiring ontology amendments.

- **Minimal encoding bias:**

  To insure this aim, we separate the ontology concepts from specific symbol encoding. Also, the definition of ontology terms should be formalised, since many knowledge agents may use different encoding systems.

- **Minimal ontological commitment:**

  An ontology should capture the minimal sufficient vocabularies for describing a domain. Nevertheless, it should give the participants freedom for customisation [48].

3.3.6 Steps in Ontology Creation

In designing an ontology we should consider that:

- Devolving ontologies should go through iterative procedures.
- Any domain does not have one correct modelling design, since specifications vary from one application to another, and the future extensions might have new requirements.

The method used for creating ontologies, as described below, consists of a set of steps, with corresponding heuristics. The ontology designer should go through a series of steps which are known as the lifecycle of ontology:

- Specify the scope and the domain of the ontology;
- Reuse concepts residing in existing ontologies;
- Identify the significant concepts of the ontology;
- Define the class hierarchy;
- Enumerate properties for each class;

- Define the facets of relationships;
- Create ontology instances.

A class hierarchy can be created using three approaches [32]. These approaches concern the method of producing classes and class hierarchy; however, the creation of properties is not considered. These approaches focus on building the taxonomy aspects of ontology classes, as the majority of ontologies designs do. The three design approaches are [64]:

i. Top-down:

   This process determines the general terms of the domain, and then creates a specification of each term. In other words, we create the most general classes, and subsequently create the subclasses. Also, this process includes a categorisation for each subclass. Here each ontology designer tries to create the best classification to represent the application needs from his perspective.

ii. Bottom-up:

   This process defines the most specific concepts, followed by the grouping of these concepts into more general concepts. In other words, the leave classes of the hierarchy are defined, then generalised into a common superclass which could be a subclass of another superclass and so on.

iii. Combination:

   This process is based on the idea of integrating both bottom-up and top-down approaches. First, it defines the most significant concepts, and afterwards applies taxonomy methods by generalising or specialising these defined concepts: for example, dividing classes into three levels; top level, middle level, and specific level. Finally, we relate top level classes and specific level classes through middle level classes.

None of these three approaches is superior to the others. Rather, the choice of approach depends on the designer's personal view of the domain: if the designer establishes the design by the grounded specific terms then the bottom-up methodology is more

appropriate, whereas if the designer has a system where the most general concepts are first in the hierarchy, then the top-down methodology is better. Some researchers claim that the easiest way to develop an ontology is by using the combination approach, since the middle concepts are more descriptive in the domain. In our method we chose the top-down approach, because the concepts already existed from the entities in the database model.

3.3.7 Challenges in Building Ontologies

There are many challenges in building an ontology. These challenges might affect global or local ontology, or both.

The main challenge is the completeness of the ontology design, which means making sure all the relevant concepts in the domain are included. There are many suggestions for overcoming this challenge. The first is to enlist the participation of a domain ontology expert, especially for designing the global ontology. However, in the case of local ontology, the conceptual design of the database is sufficient and there is no need for a domain expert; any developer familiar with the domain is suitable. The second suggestion is to identify the most frequent concepts, which involves applying this technique to different real cases. Using real cases and analysing them will help the designer to infer all the related concepts of the domain. The third suggestion is to identify the synonyms of each concept with the help of a thesaurus e.g. WordNet: this will help in limiting the problem.

The second challenge is ontology evolution, which refers to the changes in ontology over time. These changes include the adding, modifying, and deleting of some concepts. Since the domain itself evolves over time, the design of the ontology will be affected.

The third challenge is the risk that the domain is too large. Hence the expert should determine the scope of the domain. This can be done by breaking down the domain into sub-domains; then for each sub-domain a local ontology can be created. The idea of local ontologies is to help manage the relationships between concepts. Further to this, local ontologies can be mapped to a global ontology which shows the whole domain. Another way to provide a view of the whole domain is to integrate local ontologies with

the global ontology. With either method, the domain expert should ensure that the relationship between concepts is consistent.

3.3.8 Ontology Description Languages

In fact, ontology languages are considered to be the foundation of knowledge-based systems. Also, they can express knowledge specification in a rich and intuitive way, and in a machine understandable form.

3.3.8.1 Resource Description Framework (RDF)

RDF language [9, 19, and 22] is a standard Semantic Web language which provides a description for Web resources; also, it is frequently used to represent data, properties, and Web resources content in order to exchange knowledge over the Web. It has been developed in a machine-understandable way in order to achieve interoperability between applications.

RDF is recommended by the W3C. It utilises the URIs in identifying resources, and it adopts the XML syntax. However, RDF has been designed for representing semantics.

Many applications utilise the RDF language; for instance, the improvement of search engines qualification stemmed from RDF's ability in discovering Web resources. Another example is the intelligent software agents which facilitate RDF in exchanging and sharing knowledge.

The syntax of RDF model is built on three elements: subject, predicate, and object. In fact, the subject is any resource that can be defined by URI. The RDF statement would have the following interpretation, which is <subject> has a property <predicate> valued by <object>.

3.3.8.2 Resource Description Framework Schema (RDFS)

This ontology language adopts XML syntax and is built on top of an RDF model. The RDF Schema provides additional vocabularies to the core of RDF in order to gain more expression.

RDF Schema is effective in defining vocabularies required by applications. In fact it is a set of RDF resources which express properties exists in other RDF resources in machine-understandable format. Moreover, the RDF Schema expresses descriptions for classes and properties, as well as determining specifications for relations between properties and classes.

In general, RDFS may be identified by the URI reference of http://www.w3.org/2000/01/rdf-schema#, as well as a defined namespace 'rdfs', whereas the RDF namespace is 'rdf' and its URI reference is http://www.w3.org/1999/02/22-rdf-syntax-ns#.

Class, property and property constraint are the three elements of ontology which can be described by RDF /RDFS.

RDF is a good basic language foundation which can be utilised in building other languages. However, it has a limited ability in resource description such as cardinality, domain and range localised constraints, existence descriptions, and property characteristics (transitive, symmetrical, and inverse). For example, 'male' and 'female' are two subclasses of the human class, and separating them from each other is impossible since RDF lacks the disjointed vocabulary. Therefore, we consider the expressive power of RDF to be limited, as mentioned in [7]. To overcome these limitations, RDFS was invented. RDFS can be utilised in building class hierarchies (subclasses) and property hierarchies (sub-properties); also, it enables relationship construction and restricts domain and range, besides providing instances for classes

However, RDFS is not sufficiently expressive in further aspects. It is unable to describe equality and inequality between properties. Also, it cannot define enumeration. Union, intersection, unique, symmetric, transitive and inverse are relation characteristics which cannot be expressed by RDFS. Consequently, many ontology languages, such as DAML+OIL, and OWL, have been developed to tackle these deficiencies.

3.3.8.3 Ontology Interchange Language (OIL)

Ontology Interchange Language: OIL is a markup language built upon RDFS. One of its important features is providing modelling primitives which have been utilised by frame-based ontologies [15].

3.3.8.4 Annotated DAML+OIL Ontology Markup

DARPA Agent Markup Language (DAML) + Ontology Inference Layer (OIL), or DAML+OIL [17, 24] is a semantic markup language. It is an extension of RDF which tries to reduce some of the RDF deficiencies through building richer primitives. Also, it has the ability to provide a description for domain structure in terms of classes and properties. One of its powerful characteristics is that it can formalise the meaning of language terms by applying the theory of a Description Logic (DL) model [21].

The production of DAML+OIL is to amalgamate European languages with the American proposal.

Like other Semantic Web languages, DAML+OIL has its own limitations, one of which is the lack of property descriptions; for example, DAML+OIL does not offer composition or transitive closure, and does not allow comparison between data values. Moreover, it represents collection type by sets only; i.e. it does not offer lists or bags.

3.3.8.5 Web Ontology Language (OWL)

OWL language [7, 13, 23, 29, 30, 51], is a standard Semantic Web language for processing information over the Web, recommended by W3C. It facilitates RDF vocabularies and XML syntax. However, it overcomes drawbacks appearing in other languages such as RDFS and DAML+OIL. It also has the ability to provide an interoperable data environment for different domains and communities.

If a comparison takes a place between OWL and RDF, we would find some similarity between them; however, OWL shows more capabilities with regard to semantic language vocabulary which enables it to provide machine interpretability. Evidently,

OWL vocabularies have richer semantics than RDF and RDFS in representing property characteristics and the hierarchy of classes and properties.

Overall, OWL was invented to utilise XML syntax and to adopt RDF and RDFS primitives; for example, it uses RDF terms and meaning in defining classes and properties. It was also designed to overcome RDF weaknesses.

OWL language is classified into three sublanguages, with various aspects to supply different goals: OWL Lite, OWL Description Logic (OWL DL) and OWL Full [28].

The simple version of OWL is OWL Lite which provides classifying hierarchies and forms the constraints in a simple way. It expresses only maximum cardinality of relationships in a value of 0 or 1, and thus produces an easy design. The restriction is limited in OWL Lite, since it lacks some terms such as 'union' and 'negation'. Therefore, it has the lowest expressiveness of OWL sublanguages. OWL Lite is considered as a sublanguage of OWL DL.

OWL DL utilises Description Logic in describing relations between objects and their properties. It tries to preserve the completeness of computational properties; therefore it is most expressive in describing concepts and relationships.

The sublanguage OWL Full provides the most expressiveness and the syntactic freedom of RDF but without preserving guarantees on computational complexity. OWL Lite and OWL DL are sublanguages of OWL Full.

- **OWL Syntax**

OWL has two kinds of syntactic representation forms. The first one, which Chapter 6 uses this syntax throughout the examples, is known as the exchange syntax. This form represents ontology as a set of XML or RDF triple which is ready to be published and shared over the Web. However, the exchange syntax form is difficult to trace since it contains a large number of syntactic constructs for each class or property. The second form of OWL syntactic forms is the abstract syntax. This form is abstracted from the exchange syntax and appears similar to relational schema. One of the characteristics of this form is the frame-like style which constructs all the information about class or

property in one collection. Thus it facilitates the assessment steps of the produced ontologies. The evaluation chapter utilises this syntax.

Table 3.1 shows a comparison between the above discussed ontology languages, as well as the limitations of RDF, DMAL+OIL in contrast to OWL language. Therefore, OWL is considered to be the most expressive among Semantic Web languages.

The expression	RDF/RDFS	DAML+OIL	OWL
Class	√	√	√
rdf: Property	√	√	√
rdfs: subClassOf	√	√	√
Rdfs: subPropertyOf	√	√	√
Rdfs: domain	√	√	√
Rdfs: range	√	√	√
Individual	x	√	√
Same Class As	x	√	√
Same Property As	X	√	√
Same Individual As	X	√	√
Different Individual From	X	√	√
Inverse Of	X	√	√
Transitive Property	X	√	√
Symmetric Property	X	√	√
Functional Property	X	√	√
Inverse Functional Property	X	√	√
All Values From	X	ToClass	√
Some Values From	X	hasClass	√
Min Cardinality	√	√	√
Max Cardinality	√	√	√

Table 3-1: Comparison between ontology languages

3.3.9 Ontology Applications

To build a common understanding and consensus in knowledge areas, ontologies have become a ‘hot topic’ of research. Ontology first appeared in Artificial intelligence (AI) laboratories, before being used in other fields. The following are examples of ontology use:

I. **Semantic Web** [8]: Ontology is the key enabling technology in the Semantic Web to support information exchange across distributed environments. It meets

the promise of the Semantic Web in representing data in a machine-processable way.

II. **Semantic Web Service Discovery** [8]: Ontology helps in describing the merchandise in the E-Business services, which also includes trying to discover the most suitable match for the requester obtained and establishing a communication between buyer and seller. These facilities are also applicable in an E-Commerce environment [16].

III. **Artificial Intelligence** [26]: The AI research community was first to develop the idea of ontology. The intended goal of ontology is to produce sharing and reusable facilities for knowledge that will ensure communication between services, programs, or organisations across a given domain.

IV. **Multi-agent** [16]: in order to ensure the success of agent communications, there is a need for common understanding for domain knowledge and shared vocabularies. These requirements are offered in ontology models and any misunderstanding is thereby reduced.

V. **Search Engines** [16, 28]: Ontologies help Internet search engines in capturing common terms and defining synonyms for terms which usually exist in thesauri.

VI. **Interoperability** [12]: In the Web, systems have two aspects, heterogeneous and distributed, and are thus called the 'interoperability problem'. In order to resolve this problem, ontology can be used to facilitate interaction between heterogeneous systems by explicating the concept terms of each system in the form of a sharable and machine-understandable ontology.

VII. **Systems Engineering:** Ontology can be exploited in defining system requirements, and even for the developed systems; it helps in determining the extension purpose or maintenance specifications. Further, it can be used to resolve inconsistency in any system design.

3.4 Summary

This chapter introduced the topic of Semantic Web and its importance in upgrading the current Web. Moreover, it discussed the Semantic Web layer. After that, ontology definition was presented. The elements that participate in building the ontology were elaborated. Also the formal structure of ontology was discussed. Then we described the

criteria that ensure the success of ontology design and the challenges which need to be overcome. After that, the three methods of building ontology, top-down, bottom-up and combination were mentioned. Then we showed that the preferable method is the top-down and how our approach will utilise it in designing an ontology. There are many ontology languages; this chapter has explained each language advantages and disadvantages. Finally, we chose the OWL language for our approach since it overcomes the defects appeared in other ontology languages.

CHAPTER FOUR: OVERVIEW OF EXISTING APPROACHES FOR TRANSFORMING DATABASE TO ONTOLOGY

Objectives

- Examine existing transformation systems and classify them.
- Detail the advantages and disadvantages of each approach.

4.1 Introduction

The early work undertaken on extracting semantics out of database schema by reverse engineering methodologies dealt with converting the relational model to the Object-Oriented model [33, 34]. However, this work is unlike the approaches of transforming relational databases to ontologies because there are many missing semantics required to construct a full ontology which are not obtained by these methods.

Since ontology is able to search and share data drawn from various sources, it is the ideal Semantic Web solution for database integration. Generally, there are two global methods to integrate databases into the Semantic Web. The most researched method is the mapping between a database and an existing domain ontology through a wrapper [45]. The drawback of wrapper systems is their labour-intensity, but little work has been done towards automating them. The second method is more concerned with building automatic transformation from database schema to produce Semantic Web representations such as RDF and OWL ontologies. This method is the focus of our research.

There are different approaches which utilise the idea of this second method; the reverse engineering target for a relational database is ontologies for these approaches. Based on the centred source type, they are roughly classified into one of the five categories given below:

I. Approaches based on an analysis of relational schema.
II. Approaches based on an analysis of tuples.
III. Approaches based on HTML pages.
IV. Approaches based on Entity Relationship (ER) or Extended Entity Relationship models (EER).
V. Approaches based on Structure Query Language (SQL).

This chapter will explain each category through presentation of one or more detailed examples; first with a brief description and then by detailing the advantages and disadvantages of each category.

4.2 Approaches based on the analysis of relational schema

In this category, all the approaches use relational schema as the database source. Here, we differentiate between database schema and SQL as the source. Database schema is more theoretical, whereas SQL is for the actual implementation of the schema. However, approaches in this category utilise some characteristics from both models.

4.2.1 The Approach used by Stojanovic *et al.*

This is the first work produced with the intention of writing transformation rules from relational database schema to ontology [36]. This approach sought to transform relational databases to Frame Logic (F-Logic) and then represent the F-Logic model by RDFS. Using F-logic can be considered simultaneously both advantageous and disadvantageous. The advantages of F-logic are:

- There are many components which draw on the Frame Logic language; these include object identity, complex objects, inheritance, polymorphic types, methods, and encapsulation; also, the language has the ability to integrate functions into a logic-based framework.
- The higher-order of Frame Logic syntax enables both data and schema to be integrated in one view.
- The same declarative language is used for manipulating data and schema together.

Whilst the disadvantages of using F-logic are:

- Frame Logic does not distinguish between different basic types; it treats all of them as objects.
- Attributes and associations are not distinguishable in F-Logic.
- F-Logic uses parameters to model N-ary relationships, but this feature cannot be utilised as RDF cannot express it.

The Disadvantages of the Approach used by Stojanovic *et al.*

The main problem with this approach is using a middle process; i.e. it requires the generation of two models. The first model is in F-logic then this is mapped to RDF Schema ontology. The second problem of this approach is targeting RDF Schema ontology which has many drawbacks such as:

- Modelling primitives of RDF Schema lack formal semantics and use of these primitives may cause an interpretation problem which might be used in an error-prone process.
- Targeting RDF ontology forces the approach to introduce a new class for each basic type.
- RDF Schema does not have an inference model.

The third problem is that the approach mentions the importance of evaluating and validating the process however it did not apply them. In addition, although the approach is automated, the generation of the ontology requires the administration and revision of the designer with some steps relying on user interactions. Additionally, the approach shortcoming is apparent since it neglects database evolvement and data updates. The approach is designed to admit SQL statements as input, but in reality, it is based on database schema.

Some general drawbacks in the transformation rules of this approach include:

- The user decides between a specialization relationship or a fragmentation relationship and makes the decision depending on individual experience.
- There is no distinction between strong and weak entities.

- The approach does not consider the case of the multivalued table.
- The approach does not consider an N: M relationship with attributes.
- The approach does not solve the problem of *N*- ary relationship tables.

4.2.2 The Approach used by Li *et al*.

This approach [41] looks at acquiring OWL ontology automatically from databases in relational models by applying a group of learning rules instead of using a middle model. One of the characteristics of this approach is that it involves obtaining a complete OWL ontology, which includes; classes, properties, property characteristics, cardinality and instances. The second feature of this approach is that the resultant ontology is produced in OWL language. OWL language is enriched with formal semantics vocabularies which enable it to facilitate better machine understanding for Web content than the earlier languages such as XML, RDF and RDF Schema.

This approach requires knowledge of some SQL features; e.g. primary key, foreign key, UNIQUE, NOT NULL which help in building axioms.

The Disadvantages of the Approach used by Li *et al*.

An appraisal of their rules found these negative aspects:

- The first learning class rule of integrating the information, which is spread across several relations (vertical fragmentation) into one ontological class is confused because the condition rule can be satisfied for the specialisation (inheritance) situation as well. This means the condition does not distinguish between fragmentation and inheritance. As a result, this rule will confuse a reader that is applying it manually.
- The rules have many deficiencies, an example of which can be found in the second learning class rule which contains these conditions:

  "Create a class if one of the following conditions can be satisfied and the first rule cannot be applied:

  i. $|pkey(R_i)| = 1$;

ii. $|pkey(R_i)| > 1$, and there exists A_i, where $A_i \in pkey(R_i)$ and $A_i \notin fkey(R_i)$".

This rule presents the following problems:

- The first part of the condition, which compares the number of primary keys in the database table to the number one, is incorrect. As is known, there is only one primary key for each table. As the primary key can contain a simple or composite attribute, the condition must specify the number of primary key attributes in order to be accurate. Furthermore, there is no need to count the number of primary key attributes since the rule concerns whether or not the foreign key is part of the primary key. In other words, there is no inclusion dependency based on the primary key.
- The authors move between formal notation and English description in their rules when they should be written in formal notation only. For instance, the part of the condition "Rule 1 cannot be satisfied" described in English could be rewritten to exclude all tables which satisfy fragmentation or inheritance as below.

  $$|pkey(R_i)| = 1 \text{ , and } pkey(R_i) \notin fkey(R_i).$$

  Here we present the argument of what benefits will be gained if the system rules utilise formality. And the answer would be: if the rules written in English description or mix formal notation with English description; it might lead to an ambiguous interpretation. Therefore formal rules will have a precise meaning and clear understanding. Also, formality unified the rules interpretation.

- The rule of instances is not applicable for all different instances of classes. At this point, it is obvious they only used a very general rule. However, the instances rule must include different procedures of learning instances from strong entity, weak entity, *N*-ary relationship or multivalued attribute and must avoid using a very general rule.

To summarise, the problems associated with this approach are:

- The rules of Li *et al.* are not defined in a full formal notation; however, it is a combination of English language and formal notation. We believe the shortcomings of this approach are due to this lack of a formal system.
- The rules are designed for their specific example of University database and not generalised to deal with different situations which could result in concepts being inaccurately depicted across domains or database modelling choices. Therefore, the approach must be examined through different examples to capture the variety of modelling choices in various domains.
- Handling the *N*-ary relation by using only object properties will not ensure that the group of binary relations exists as a whole.
- The approach does not consider the case of a weak entity, multivalued attribute or relationship with attribute.
- The approach includes instances with the target ontology, which detrimentally necessitates the ontology to be recomputed each time we change the data in the database. The solution proposed by Stojanovic *et al.* is the separation of the ontology and the knowledge base (ontology with instances) which allows the knowledge base, if needed, to be computed on demand.

There are some semantics offered by the relational schema, such as constraints on foreign keys, which are missing in their rules.

4.2.3 Other Database Schema Approaches

The next section will briefly explain some of the approaches which use the schema method.

- **Dogan & Islamaj** [37] utilise reverse engineering methods to provide a simple and fully automatic approach which contains mapping relations to classes: a relation's attributes to the datatype properties of a class; the relationship to object properties; and records of tables to individuals. However, this approach neglects the significant

of inheritance in ontology design, which results in producing a flat ontology or an ontology which has an appearance similar to that of a 'relational' model.

- **Rubin *et al.*** [38] propose an approach for extracting instances and attributes values from relational database source, and then filling these data into ontology individuals. One of the characteristics of this approach is the use of an interface implemented in an XML schema in order to declare the relationship between the database source and the ontology. However, this approach requires many components beside the XML schema interface; for example, an ontology template and an XML translator require to be automated.
- **Kashyap** [35] analysed a database schema in order to capture semantics; then he improved the result by user queries, and finally constructed an ontology based on the refined result. However, this approach misses database constraints which build an ontology without axioms.
- **Astrova *et al.*** provided an automatic system which transforms relational schema to OWL Full. This system is based on explanatory informal rules. There are several shortcomings, since the approach goal is OWL Full ontology which would not be able to be reasoned. Besides, the rules lack formality; many of them are based on specific examples which cannot be generalised and might lead to ambiguous rules [46].

4.3 Approaches based on an analysis of tuples

Acquiring database schema from the content of database tuples is the main inspiration behind approaches which fall into this category.

4.3.1 The Approach used by Sonia *et al.* [43]

The R2O is a system that aims to analyse database content in order to build an artificial relational schema in the absence of a metadata code of database. This approach considers the database content to be the source to produce the conceptual model of a database. Then it transforms the database conceptual model to an OWL ontology. Moreover, it can handle the N-ary relationship; M: N relationship with additional attributes and multivalued attributes.

The Disadvantages of the Approach used by Sonia *et al.*

Overall, the weaknesses in this approach are:

- The main disadvantage of this approach is its dependence on metadata which is changeable. Any new records added to the database can damage the consistency of the derived characteristics.
- It does not specify how to obtain database relations.
- It does not consider the relation of multivalued attributes in the analyses in the first phase. However it builds a transforming rule in the second phase.
- The authors claim that their approach relies on the feature of extracting the metadata from data sources in absence of structure of the relation and only depending on the content. However the approach required the NOT NULL constraint to be given for attributes, which is an SQL expression.
- The approach does not consider that the problem of generalization/specialization and fragmentation where both have the same conditions.

After appraising their rules we find there are many limitations in the approach to transforming rules as follows:

- The rule of specifying the secondary relation which represents the M: N relationship and the N-ary relationship has the notation of PK=FK ∧ PK=X. Where X denotes the entire attributes of a relation. We can observe from the notation that the rule does not consider any relationship with additional attributes.
- The rule of sub-type relation does not distinguish between superclass-subclass and fragmentation.
- One of the most common mistakes in this approach is the erroneous ordering of rules that may not have considered that the rule of creating a primary relation must take precedence over the rule of weak primary relations. The reason for that is the weak primary relation needs the primary key of primary relation to be included to its key.

- Each rule has included generating the classes, the datatype, the object properties and the axioms. However we cannot generate an object property between two classes where one of them might not be created.
- The approach states incorrectly that for all unary relationships, create an object property with symmetric property. We shall discuss this idea in more detail in section 6.4.4.4.
- The rules are mostly English description with some notations which might mislead the reader. In addition, the rules are not formally defined.
- The approach does not consider merging relations of fragmentation as stated in the authors' conclusion.

4.3.2 The Approach used by Astrova *et al.*

This approach [39] was originally based on relational schema. However the approach claims that the relational schema has few clear explicit semantics; therefore there is a necessity to detect the obscure semantics (e.g. inheritance), which can be captured by analysing relational database tuples. However, this approach has a varying time based on the number of tuples that require analysing in order to create a database conceptual model. Also, the success of this approach relies on the assumption that changing the content would not change the semantics derived from this content.

4.4 Approaches based on HTML pages

Most databases available online have HTML pages, which when analysed will reproduce or enrich database schema. This is the core idea behind the approaches classified in this category.

4.4.1 The Approach used by Benslimane *et al.*

The authors utilise a reverse engineering method in a semi-automatic system which obtains OWL ontology from relational databases. The approach [42] capture semantics hidden in HTML forms then use the extracted information to rebuild or to augment the

relational schema. After that, the enriched schema will be the source description for their transformation system in order to produce OWL ontology.

The main features of this approach are:

- It can be useful in a situation where the owner of the database is uncooperative.
- The approach considers using a semi-automatic approach as a feasible solution for producing ontologies from relational database, since the fully automated approaches suffer from the problem of missing some semantics which cannot be obtained by automatic analysis.
- The procedure of transforming the XML schema to a hierarchical schema can help in identifying the class-subclass relationship which has a major influence in the structure of ontologies.
- Constructing OWL ontology from the enriched relational schema using a set of transformation rules without demanding a middle model.

The Disadvantages of the Approach used by Benslimane *et al.*

The main disadvantages of using this approach are:

- The design of HTML pages requires regular restructuring - more than twice a year - so any changes to the structure of HTML can disrupt the constructed ontologies.
- This approach suffers from modelling overhead since producing the ontology requires going through five engines; also, the system needs to manage between three language models that are used as sources to generate the target ontology. For example, besides the relational model schema there are three schemas - Form model schema, XML schema, and the structure schema - generated during the application applying the algorithm. After being generated, they expand the relational model schema by the semantic derived from the structure schema.
- In the analysis part of the extraction engine, the rules for identifying linked attributes and structural units depend totally on visual cues. Additionally, all the rules in this engine do not include detailed instructions explaining how to apply

them. Furthermore, all rules in this engine are user dependent. All of which makes the approach tedious and time-consuming.

- The generation of the XML schema part of the extraction engine includes many rules, one of which unrealistically assumes that all filling fields are mandatory.
- Most steps generated by the extraction engine and the enrichment engine produce redundant information which can be obtained by using the relational schema only.
- The steps of extracting the functional dependency and the inclusion dependency are only applied on the Form instances; and therefore, make unclear decisions since the structure unit of the Form does not always represent a relation in actual database schema. For instance, the node of the hierarchal structure may be equivalent or similar to several relations in the underlying database.
- In this approach the main transformation rules from relational schema to an ontological model are identical to those in the approach used by Li *et al*., which we already discussed, whilst the rules of constricting the instances are already proposed by Stojanovic *et al.*
- Most steps in this approach rely on the analyst's observation to catch the semantics, and this will raise questions about whether it is easier to build ontologies from scratch or if it is preferable to make the effort to capture the semantic of a legacy system.
- The approach has many algorithms which need to be verified and evaluated.
- With rules based on English description rather than notation, readers have different opinions when making design decisions which suggests that the approach needs to be formalised.

4.4.2 Astrova's HTML Approach

Another approach using the same method of HTML analysis is Astrova's HTML approach [40], which analyses HTML-forms in order to produce a form of schema model, before transforming the extracted model into an ontology. After that, the ontology is filled with instances obtained from the database behind the HTML form. Despite using HTML as the main source, there are other drawbacks, such not offering

any method to acquire an inheritance relationship, which is considered to be one of the important aspects in the ontology development process.

4.5 Approaches based on entity relationship (ER) or Extended Entity Relationship model (EER)

The originality of the approaches which fall into this category is based on the idea that both database and ontology have a conceptual model and that translating from one conceptual model to another is easier and could preserve more semantics.

4.5.1 The Approach used by Upadhyaya *et al.* (ERONTO)

The aim of this approach [44] is to build domain ontologies in OWL from Extended ER diagrams. This reduces the developmental efforts by automating the process and by showing the differences and the similarities between the expressive capabilities of these two conceptual modelling methods.

In the process of translating EER diagrams into relational schema certain semantic information is lost. Therefore, the approach focuses on capturing conceptual semantics from an EER diagram in order to construct a corresponding ontology. The fact that both extended entity relationship and ontological models represent conceptual models makes this approach distinguishable from others. The EER model makes an important contribution since both EER and ontology models can be considered as semantic modelling. The system requires a domain expert which helps in capturing more semantic information in order to gain an enhanced ontology.

The main advantage of using this approach is that it covers different cases such as Many-To-Many relationships with additional attributes; N-ary relationship; aggregation; and multiple inheritance. More semantics information (e.g. cardinalities) are preserved in case of translating EER models into ontologies better than relational schema.

The Disadvantages of the Approach used by Upadhyaya *et al.*

There are many deficiencies regarding this approach:

- The main problem is that, in most cases, the conceptual model (e.g. EER model) is not created during the database designing process, or this model has now been lost.
- The second problem is that, even if the ER is available, most changes will be done on the active database model (SQL) without modifying the essential ER model.
- The approach mainly depends on the domain expert to analyse the extended entity relationship model and to use his/her knowledge to enhance the generated ontology.
- There are some problems in representing the entities and attributes method:

- The approach does not consider the complex attribute.
- The claim: "An attribute, which is a primary key, will be represented as a data type property, with functional and inverse functional property characteristics, together with cardinality restriction set to one" has many problems, such as:

  i. The primary key can be represented as a data type for a strong entity while, for a weak entity, the attribute(s) corresponding to the primary key of owner entity can be represented as an object property with the discriminator represented as a data type property.
  ii. Forcing the primary key to be represented as a functional data type is acceptable but the tools do not support the feature of inverse functionality for a data type.

4.5.2 The approach used by Xu *et al.* [47]

This approach derives its strength from formal rules which translate an ER model into OWL ontology. It provides an automated tool that can read and analyse the ER schema script coded in XML, produced by CASE tools such as Power-Designer. Then the approach employs its mapping rules in order to translate the ER schema to OWL-DL in both OWL abstract -syntax and exchange syntax (RDF/XML syntax).

The approach has many drawbacks:

- Its formal method builds upon the assumption that all attributes are single-valued and mandatory, which is not the case in reality.
- It takes into account the simple-attribute keys only; however, it ignores the composed key formed from composite attributes.
- It does not consider composite or complex attributes where they available in the ER model.
- It proposes rules for transforming ER to OWL (Abstract Syntax) then to produce OWL (exchange Syntax), in order to reach the target ontology.
- The tool, Power-Designer, used in the implementation has limitations as it currently neither supports N-ary relationships where N>2 nor attributes on relationships; therefore, it does not support the features available in the ER diagram.

4.6 Approaches based on Structure Query Language (SQL).

Approaches from this category derive their power from the database creation language. Since each database reserves the structure of its tables in structure query language (SQL), an easy way to obtain any semantic information could be from this source.

4.6.1 The approach used by Tirmizi *et al.* [45]

The approach proposes a First-Order Logic (FOL) based system which automatically transforms the SQL-DDL schema into OWL ontology. The transformation rules are formal and stratified, which enables the integration of the system in Datalog interpreters that support database environments, or it can be easy implemented in Prolog editors.

The Disadvantages of the Approach used by Tirmizi et al.

- To remove ambiguous syntax and semantics, the First-Order Logic (FOL) is used, however it is very complicated to trace or predict the result.
- The approach disapproves of other work using domain-specific examples which can lead to incorrect rules. However, they fall at the same hurdle.

- The approach ignores the fragmentation of two or more tables representing one entity because they claim that the database will not be in the third normal form. However, there are no obvious functional dependencies that would violate 3NF in the vertical partitioning of tables. Any decomposition of a relation does not harm first normal form and second normal form and would not harm third normal form; but that is not the case in reverse. Therefore, there is no proof for excluding the case of fragmentation.
- The rule for creating classes does not consider the relation of multivalued attributes.
- The rule also states that if an object property is functional, then its inverse is *inverse functional*, represented by the function *IFP*; but this argument is not true, since OWL specification states that if a property is inverse functional then the inverse of the property is functional but the reverse is not so.
- There are no rules to migrate the instances.
- There are many problems with data type rules such as:

  i. Since the approach does not consider the multivalued attribute all attributes created as datatype properties are functional.
  ii. Not considering the attribute with UNIQUE characteristics.
  iii. In the part of completeness, proof that the relation has more than two foreign keys does not imply that this relation represents *N*-ary as the approach stated. The condition here is whether the foreign keys are part of the primary key or not.

4.6.2 The approach used by Astrova *et al.*

This approach [46] proposes an automatic system that is capable of transforming a relational database model written in SQL to ontologies written in OWL Full. The approach based on explanatory rules influenced from real world examples. The approach also considers a Many-To-Many relationship with additional attributes.

The Disadvantages of the Approach used by Astrova *et al*.

The approach suffers from many shortcomings. Neglecting the formal definition would lead to ambiguous transformation rules. The approach has many limitations as it does not consider multivalued tables or fragmentation tables.

Using general rules, examples of which are mapping all the One-To-One relationships to inheritance, or making all unary relationships represent a symmetric data type, weakens the approach.

4.7 Summary

After examining the existing approaches that tried to transform relational database to ontology, we classified them based on their database source to five categories. The sources for databases can be relational schema, SQL-DDL, ER, EER, or HTML. In each category we explained the shortages of the source itself. Then the popular approaches of each category were discussed in term of advantages and disadvantages. After that we reach to a conclusion that the best source to represent database model is the logical model written in SQL-DDL. And to overcome this source shortcoming, we utilise the conceptual model (EER) in order to validate and enhance the results from the SQL-DDL source.

CHAPTER FIVE: APPROACH ARCHITECTURE

Objectives

- Show the general framework for database integration.
- Demonstrate the architecture for our approach.
- Explain the disparities between the ontology model and the database model.
- Defines the significant feature and main components essential for a transformation system.
- Defines the success criteria.

5.1 Introduction

This chapter consists of two parts. The first part discusses the architecture that we have proposed to solve the database integration problem, followed by an introduction to our system design, which concentrates on a transformation system design, and a brief recommendation for a design for alignment and mapping between ontologies. The second part of the chapter focuses on unifying the theory between database models and ontology models in order to successfully produce the most efficient ontology.

5.2 Model Framework

Databases hold much of the data on the Web currently despite database not being a knowledge representation language. In contrast, ontology would be a practical provider of data representation language as it has the capability to capture semantic offered by the domain. However ontology does not have database's ability in storing and retrieving data. The architecture proposed to solve the database integration problem is based on the concept of Semantic Web. Therefore in order to accomplish our aim of database integration we should integrate database and Semantic Web systems; and since ontology is the core of Semantic Web we can utilise it in semantic integration.

There are three approaches for ontology–based integration, the single ontology approach, the multiple ontology approach, and the hybrid ontology approach [64]. We avoid using the single ontology approach since it requires all database sources to be integrated to provide a unified view of a domain. The other approach avoided is the multiple ontology approach which lacks use of a common vocabulary level, which is responsible for resolving the problems of semantic heterogeneity. Our architecture is inspired by the hybrid ontology approach since the hybrid ontology approach is characterised by the following criteria:

- Reasonable implementation effort.
- Support heterogeneity view.
- In case of adding or moving a source it can provide alternative source ontology.
- Mapping between multiple ontologies is easier since ontologies share a common vocabulary.

Figure 5.1 shows the proposed framework formed in three layers: the bottom layer contains the database sources; the middle layer consists of local ontologies corresponding to the sources; and the top layer is the global ontology. It is necessary to combine three approaches to fulfil the emerging requirements of database integration. The approaches are the mapping approach, the alignment approach, and the transformation approach.

In fact, global ontology facilitates the common understanding of a structure in sharing information among ontologies from different domains. We utilise these advantages by mapping local ontologies to global ontology. In the case of local ontology, we benefit from its ability to exchange information across distributed environments to build an alignment methodology between local ontologies. However in order to guarantee the success of our proposal framework we need a comprehensive system which can be responsible for producing local ontologies that reflect the data stored in relational databases.

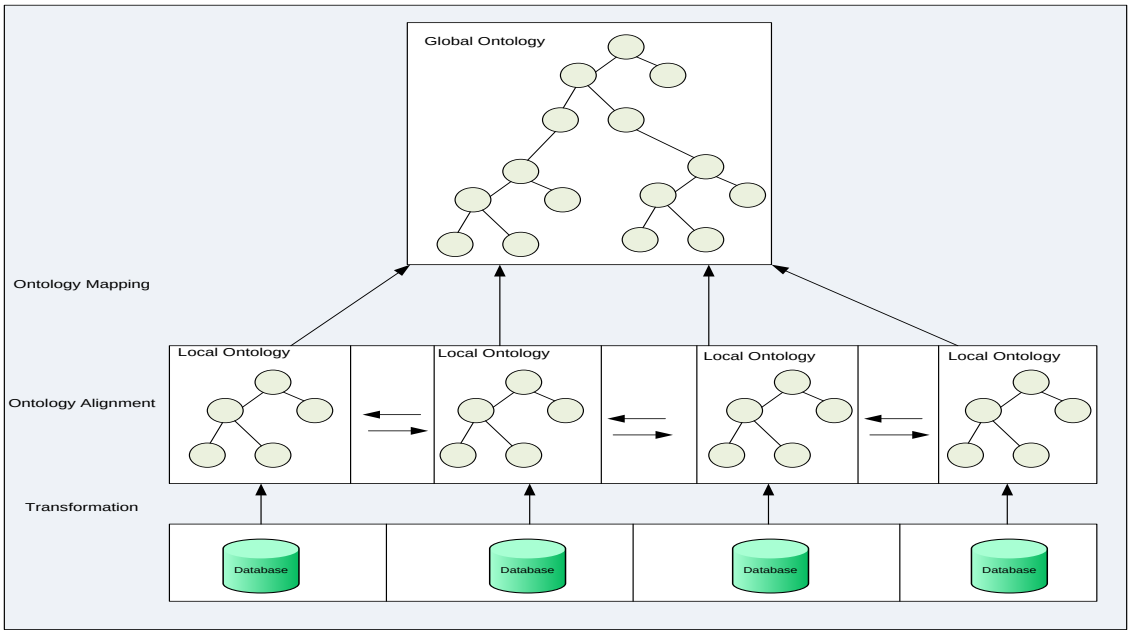


Figure 5-1: Database Integration using Ontology vision framework adapted from [64] and modified.

Notice that the framework utilises global ontology instead of global schema, since global ontology is more appropriate as far as domain modelling is concerned. In addition, ontology is more expressive when compared to relational schema. Moreover, ontology can capture more semantic than relational global schema, e.g. relational global schema lacks the ability to express the property characteristics such as symmetry and transitiveness.

To ensure the success of this framework the system needs to fulfil these criteria

- Be a scalable system.
- Have limited maintenance overheads.

This methodology aims to provide methods for producing and managing ontologies from a database design, since it exploits the semantics and the constraints of the domain applications captured through databases. Consequently, this methodology will yield to an affluent modelling approach. The outcome of this methodology is an accurate ontology which represents the world events, and is created with less effort on the part of the designer. In addition, it will enable databases to evolve and to be utilised and reused. Although our system of ontologies is not fully automatic, it captures most semantics

hidden in databases in a systematic way. Besides that, it enables us to manage ontologies less manually and to reduce the time-consuming factor.

5.3 Database integration phases

We assume complete cooperation from the database owner who seeks to integrate his database with others. The framework of our approach is depicted in Figure 5.2.

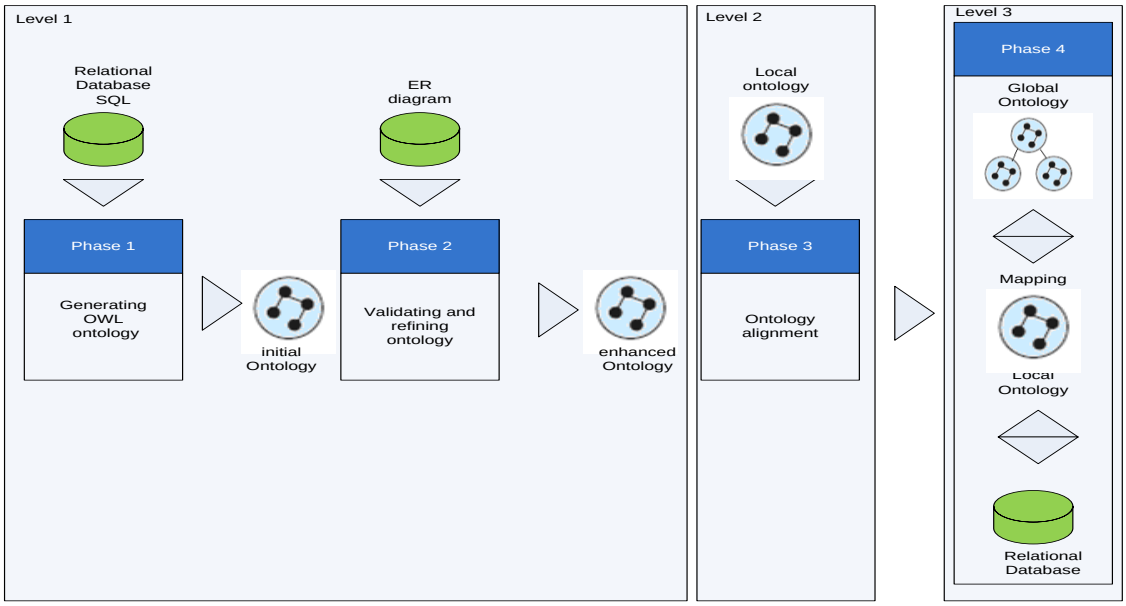


Figure 5-2: The Visionary phases Architecture for Database Integration

Our proposed system is divided into four phases:

- Generating the OWL ontology from SQL statements.
- Validating and refining the produced ontology via a comparison with the EER diagram; these two phases are our primary focus in this thesis.
- Aligning the produced ontology with the local ontologies.
- Linking the global ontology and the local ontology with database instances through a query language (we will address the third and fourth phases in future work).

Our system therefore needs to be provided with two database inputs in order to generate an ontology reflecting the database schema. The first input is the SQL-DDL, which represents the database structure, and the second is SQL-DML, which holds database instances. We can obtain this information from the database dump process, which is a way of finding the structure of the table and the INSERT data in the tuples statement. This input is central to process the first phase, after which our system applies the rules explained in chapter 6 to generate an initial OWL ontology. This initial ontology is complete from the structure design perspective since it contains all the ontology definitions (classes, datatypes, and object properties). The second input is therefore the EER model, which is responsible for validating and enriching the initial ontology produced by phase one, in order to reach an optimal result. However, the second phase involves two options; first to produce the ontology from scratch as an alternative solution, and second to utilise the EER model in the validation and enrichment process regarding the ontology generated in phase one. Here the resulting OWL ontology from phase one also acts as an input in phase two (Chapter 7); the goal from this phase is to reach an optimal result with the help of a domain expert. In the third and fourth phase, an expert will choose appropriate local and global ontologies with which to integrate the database.

5.3.1 Transformation System Architecture

Figure 5.3 explains in detail the first and second phases. There are five steps to produce the ontology in our approach. The classes-datatype creation and the object properties creation deal with the relational model of the database, more precisely SQL-DDL; the result is an initial OWL ontology. Next there is ontology validation, followed by the enrichment of the ontology. These last two steps deal with both the EER model and the ontology produced by the first two steps. Finally there is the data migration step, which populates the refined ontology with instances obtained from SQL-DML in order to generate a knowledge base.

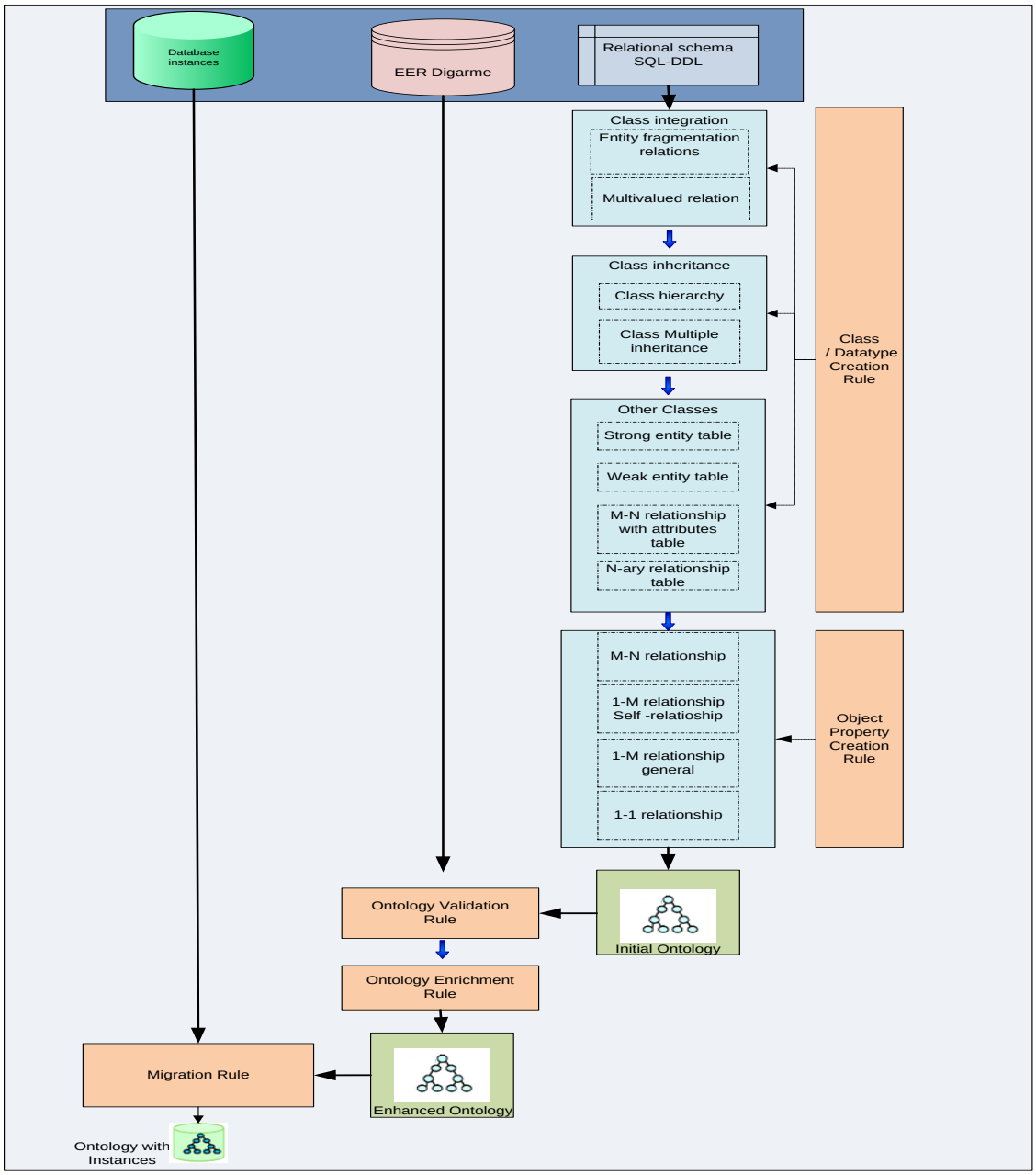


Figure 5-3: Database Transformation Architecture

5.3.2 Ontology Alignment

The second and third phases can be accomplished through a combination of techniques suggested by [62]:

- **String Matching** is used to compute similarities based on the names of class and property. It can also be used to compute the similarity between two classes by computing the similarities between the names of their properties.
- **Linguistic-based Strategies** match the strings with a thesaurus in order to obtain synonyms and hyponyms.
- **Structural Matching** is a method used to compute the similarities between two classes using graph information, i.e. computing the similarities between the super-classes of the two classes.
- **Heuristic-based Strategies** combine several features of the string matcher with those of iterations, computing the similarities in order to achieve high-quality results (see Figure 5.4).

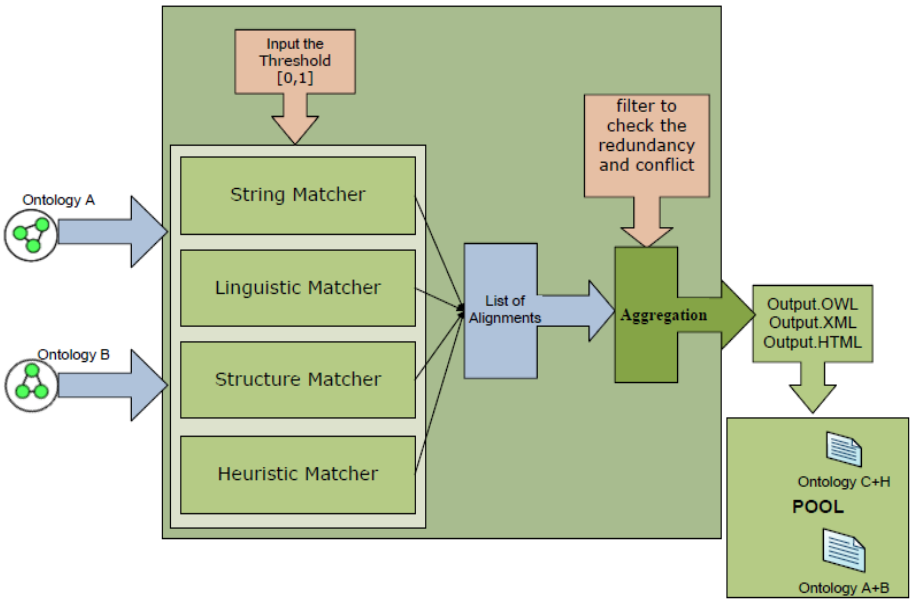


Figure 5-4: Mapping Techniques between Ontologies

In this thesis we concentrate only on the first two of the database integration phases, leaving the third and fourth phases for future work.

There are many important points to be addressed:

- Typically database integration systems contain local databases and one global schema. In our approach we use ontologies instead of global schema, which includes local ontologies and a global ontology.
- Databases convert to local ontologies which can have instances.
- The global schema will be represented by a global ontology.

5.4 General Disparities between Relational Databases and Ontologies

Here we discuss some key matters that may affect our transformation system, which focus on disparities between the database and the ontology models. The first matter is the aim of modelling, and object to model for the two models, and the second is the open/closed world assumptions for each model.

5.4.1 Aim of Modelling and Object to Model

According to Fonseca [57] there are two principles that allow us to distinguish between ontologies and database models; the aims of modelling and the objects to model. Using the first principle, ontologies concentrate on the description of the constant characteristics, whereas database models link between the domain-constant characteristics with a set of specifications defined within problem space. Using the second principle, objects to model, the ontology describes a domain structure which enables information integration. In contrast, the objects being modelled in the database represent individual events linked to a certain domain [56]. We can additionally distinguish between the two models from a different perspective since a relational database is designated to deal with a large amount of data; its main goal is therefore to manage this data in an efficient way by building a strong data structure, whereas the target of an ontology is to represent knowledge. In other words, databases tend to be more dynamic in representing real events, whereas ontologies are of a static trend. Accordingly, each model will require different domain specifications.

5.4.2 The Effects of Open/Closed World Assumptions

There are two kinds of assumptions about the control of constraints in any model: the closed world assumption and the open world assumption. Databases run under the closed world assumption, meaning anything not in the database is identified as false. In contrast, knowledge bases (ontologies with instances) usually run under the open world assumption, meaning knowledge not contained within the ontology is considered unknown. Some researchers therefore refer to the knowledge base as an incomplete database, with unknown knowledge that can grow incrementally. Consequently integrity constraints can mean different things according to the two different world concepts. Databases, for example, use constraints to validate the input data, whereas ontologies declare constraints to augment classes and relationships with certain characteristics for consistency checking. These constraints would not prevent declaration of any facts; however, during the inference process some declarations may produce unexpected (unintuitive) results. For example, in the Person table we add some information about people, including Social Security Number (SSN), Name and Address, etc. The primary key of this table is the SSN; therefore each person must have a unique SSN. In order to capture these characteristics we utilise constraints of the primary key which have both the NOT NULL and the UNIQUE constraints. When we try to add a new person without an SSN, the insertion step will consequently be rejected. Additionally if we manipulate a person's record by deleting his SSN, this action would not be updated - the integrity constraints in databases work as data verification to check whether or not the information satisfies certain conditions. This feature gives databases powerful capabilities to store and retrieve large-scale data. However, ontologies do not formally interpret axioms and assertions as database integrity constraints. For example when we create a Person class to represent a Person table, the SSN attribute can form a datatype property. Similarly, to express the dependency in OWL we impose the cardinality of one and the existential restriction to represent the primary key integrity constraints. However, ontologies do not force individuals to have a unique SSN and do not force the datatype not to accept a Null value. Further, they would accept the missing information and the Null value, i.e. the unknown information is accepted in knowledge bases. Therefore B. Motik, *et al.* [58] suggest omitting the integrity constraints during the

transformation since their corresponding axioms do not check the correctness of the input data; they also claim that these axioms may cause performance overload during reasoning, and they suggest OWL extension with true database-like integrity constraints. However, we suggest transforming database integrity constraints which can be utilised through the use of Semantic Web Rule Language (SWRL) languages. Some database constraints can be expressed only by triggers. Therefore, these constraints are suitable only for consistency checking, but not suited for reasoning, since this type of constraints is expressed in an imperative form.

In order to utilise integrity constraints in an ontology we suggest that the two models work together. The database takes on the role of storing, retrieving and validating data while the ontology takes the role of representing information in the knowledge representation mode. This leads us to the conclusion that an ontology is not a substitution for a database.

While generating ontologies from a relational model, we therefore have to keep in mind considerations of the above-mentioned differences between the two models. As a result we have to face the question of whether the produced ontology would belong to the closed world or not. The answer is based on the domain and the application requirements, although we generate our ontologies with an open world assumption. For example the default of database tables is disjoint, whereas OWL considers classes to be overlapping. In order to produce a typical ontology we would not close the open world; therefore we bypass the declaration of pair-wise disjoint classes' axioms.

It may be asked whether these differences between the two models, addressed by [57 and 58], affect our rules. Indeed these differences must be taken into consideration during the transformation process; however, since our target is to produce local ontologies (application ontologies) these differences would not be obstacles to our rules, since the local ontology is nearer to event modelling than to domain ontologies (global ontologies). In fact the local ontologies may even cooperate with each other as 'siblings' for their parent global ontology.

We consider the mismatch between the relational model and the ontological model in our transformation rules. In addition there are some slightly different design choices

depending upon domain requirements and model capabilities, which should receive careful attention while we transform them.

5.5 Comparison between Ontology and Conceptual Data Model

This section explores similarities and differences between ontology models and conceptual data models. There are some similarities between conceptual data models and ontologies; both share similar designs in representing concepts, for example entity sets in conceptual data models correspond to classes in ontologies. Both models also use relationships to link between concepts. Moreover EER and ontological modelling could both design the specialisation and generalisation inheritance by ISA and subclass constructs respectively. Here we have to distinguish between the two conceptual data models EER and ER, since the former supports inheritance concepts whereas the latter does not. Thus, the methodologies for developing EER models and ontology models have a lot in common [56].

While ontologies and conceptual data models share common features, they also have some differences. All the differences explained above (5.4.1 and 5.4.2) are at the concepts level, but there are also some differences at the details level.

Firstly EER has strong representation for higher-degree relationships, while ontology handles them indirectly through classes and object properties. OWL is more expressive than EER in some aspects, for example, OWL focuses on property characteristics such as value constraints, transitiveness, and symmetry, which are absent in EER. In contrast, conceptual models have focus in capturing structural constraints such as keys and cardinality constraints. The cardinality concept is represented by OWL terms, better than the EER model, since OWL are rich with expressive power terms in general. If EER uses the notation of (1 and M) cardinality then ontology will surpass it. However if EER cardinality is based on the notation of (min, max), then both models are equally good at cardinality representation.

5.6 Comparison between Ontology and Relational Data Model

Here we need to discuss the evolution of relational modelling before focusing on the disparities between the two models.

5.6.1 SQL Evolutionary Stages and Ontology Layers

Before SQL was invented, relational models only had relational schema syntax with which to consider the representation of data. SQL then took its place in representing relational models in three parts: Data Definition Language (DDL), query language, and Data Manipulation Language (DML). SQL-DDL is described as a declarative language for defining both the physical and logical representation of data in databases, which is our concern.

SQL evolved during the last two decades and has many standard versions, from SQL-86 until SQL-2008. The most important features that play a major role in SQL are:

- Table definition (SQL-86).
- Data integrity constraints (SQL-92).
- Triggers which are used to maintain correctness (SQL-1999).
- XML-related features, however these do not add any specific semantic components to the DDL features (SQL-2003).

Our approach concentrates only on table definitions and data constraints. Figure 5.5 shows SQL-DDL features. The development of SQL-DDL has an equivalent corresponding language in a semantic web layer cake. Each super feature in a semantic web layer cake offers more semantics than the previous one.

Indeed our approach does not consider the full SQL constructs for two reasons, the first being that not all database systems fulfilled the SQL standard during their implementation, and the second being that many parts of full SQL-DDL are vendor specifications. We therefore devote our approach to handling only the primitives of relational models.

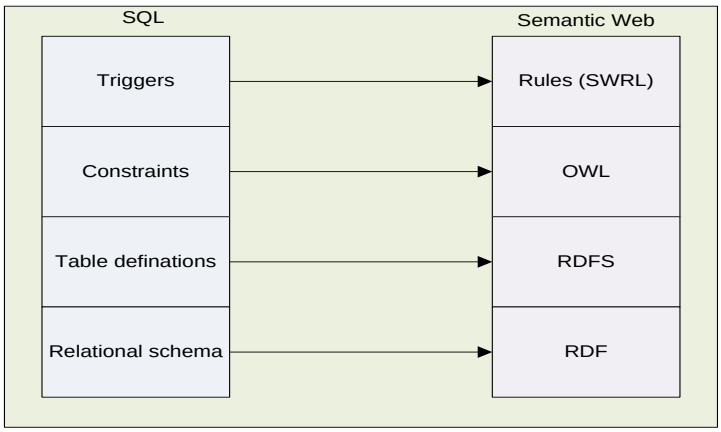


Figure 5-5: Mapping between Relational Models and Ontology Models Adapted from [61]

Consequently there are parts that cannot be mapped to a corresponding model in OWL language such as:

- SQL ordering constructs.
- Built-in functions (e.g. count, min, max... etc).
- Because of ontologies static nature, all dynamic aspects, i.e. behavioural parts (stored procedures, triggers, assertions and referential actions) cannot be translated to a corresponding ontologies terms [36].

Here also there are some common characteristics shared by both OWL ontology models and relational data models; the RM model has a mathematical base similar to the description logic (the foundation of OWL ontology).

5.6.2 General Disparities between RM and OWL

- Databases use integrity constraints to capture cardinality ratios of the relationships between entities. For example, imposing NOT NULL to a foreign key restricts the cardinality to one. However databases lack expressive definitions of relationships, i.e. there are no logical characteristics such as symmetry and transitivity etc. Ontology, however, expresses the concepts of relationships' logical characteristics naturally. More details about this are given in section (6.4.5.4). Generally information about data is not explicitly stored in a database. Sometimes it is hidden in the entity declarations; at other times, as

with triggers, it is implicitly expressed in an unclear part of a database, or in Procedural Language (PL-SQL), whereas ontology explicitly represents all the domain knowledge [59].

- One aspect that can differentiate between databases and ontologies is the relationship between the model and the real world. Usually, a database bases its design on real world enterprise analysis and draws the design from user specification, while the ontology design tends to be abstract, having an absence of specified user needs. Thus it is more appropriate to express database design in a close world assumption.
- Database design is more powerful with regard to structuring an *N*-ary relationship as a complete unit. However, an ontology handles an *N*-ary relationship indirectly through binary relationships. OWL does not provide a construct which would be able to combine the binary relationship groups in one whole unit.
- RM cannot deal indirectly with multi-valued attributes since RM requires auxiliary tables to represent it, but an ontology can present multi-valued attributes properly.

5.6.3 Inheritance Modelling Disparities between Relational Databases and Ontologies

We have dedicated this section to discussing the inheritance in both models since inheritance is crucial when representing an ontology. Therefore we will demonstrate different ways of modelling inheritance, and explore how it will affect our transformation system.

Relational models declare inheritance implicitly. Inheritance hierarchies can be identified in various ways in SQL-DDL. The problem stems from a variety of methods in modelling inheritance relationships: each modelling pattern is not restricted to represent a unique relationship; therefore the inheritance patterns could be used to represent other relationship types such as one-to-one relationships. Even using the reverse-engineering methodology would not help in deciding whether or not this relationship was originally designated for subclass relationships. It is therefore difficult

to infer the correct model choice automatically. There are two patterns to model inheritance:

- **A foreign key is a primary key too**: this is the usual design in the formation of subclass relationships. Database designers misuse this pattern, however, to represent the vertical partitioning of an entity represented by more than one table, although the pattern is considered a unique way to identify inheritance.
- **Foreign key and primary key are disjoint**: this pattern is in fact used to represent one-to-many relationships. However in this case we impose the foreign key by the NOT NULL and UNIQUE constraints to represent an inheritance. These kinds of relationships are needed to represent the shared entity, for example if an entity 'Engineer-Manger' is a subclass of a 'Salaried Employee' entity and a 'Manger' entity simultaneously. We use the first pattern for one single inheritance between 'Manager' and 'Engineer-Manager'. In modelling the second inheritance, we use this pattern to represent the multiple inheritance since there is no way to represent the shared entity by the single-inheritance pattern.

Some approaches such as [45] consider that the foreign key subset of the primary key (the ‘part of’ relationships) could model an inheritance, however we do not consider this to be the case as this pattern is usually for modelling either weak relationships or multi-valued attributes only.

Conversely ontological models exploit the subsumption relationship (subclass) to represent single and multiple inheritance explicitly in a single way.

5.7 Database and Ontology Capabilities

We now give an introduction of different database source characteristics in an ontology context. As seen in Table 5.1 there are two database sources, which include the conceptual model of EER and the logical model of RM. The table shows the limitations of each source. Since we consider the two database models as being the sources to our approach, our system can capture any ontology context as long as it is available in one of the sources.

Table 5-1: Comparison between EER, RM (SQL) and OWL Models

No	Context	EER diagram	RM	Our system	OWL
1	Concepts (human)	Possible (entity)	Possible (table)	Possible	Possible (class)
2	Roles (has child)	Possible relationship	Possible relationship	Possible	Possible Object Properties
3	Inheritance	Possible (ISA)	Not Possible	Possible	Possible (sub of)
4	Multiple inheritance	Possible	Not directly	Possible	Possible
5	Cardinality restriction	Possible	Possible	Possible	Possible
6	Value restriction	Not Possible	Possible	Possible	Possible
7	has value restriction	Not Possible	Possible	Possible	Possible
8	Transitive property	Not Possible	Not Possible	Not Possible	Possible
9	Symmetric property	Not Possible	Not Possible	Not Possible	Possible
10	Inverse property	Possible	Possible	Possible	Possible
11	Functional property	Possible	Possible	Possible	Possible
12	Inverse functional	Not Possible	Not Possible	Not Possible	Possible
13	Equivalent class/property	Not Possible	Not Possible	Not Possible	Possible
14	Enumerated property	Not Possible	Possible (check in)	Possible	Possible
15	Disjoint class	Possible	Not Possible	Possible	Possible
16	Individuals	Not Possible	Possible	Possible	Possible
17	Same/different individual	Not Possible	Not Possible	Not Possible	Possible
18	Sub properties	Possible	Not Possible	Possible	Possible
19	Keys	Possible	Possible	Not directly	Not directly
20	Ternary and higher order relations	Possible	Not directly	Not directly	Not directly

The context available in the RM model will be captured automatically whereas the context available only in the EER model can be obtained manually by the procedure suggested in Chapter 7. Using only the relational model (SQL) makes our transformation system automatic. Conversely using the conceptual database model (EER) with RM can add more semantics to the ontology, however this makes our system semi-automatic.

From the table above we can detect that if any one of the database sources (EER or RM) is capable of representing an ontological context, our approach can acquire these semantics. Generally our approach extracts the semantic from the RM source if it is accessible, otherwise the semantic will be obtained from the EER source; this ensures that there is no conflict between the results obtained by the two sources. Consequently, the need to merge the two sources results no longer exists, since the semantics obtained from the RM model are different from those obtained from the EER model.

The table also shows the limitations of our system. For example there are many different modelling contexts which could not be obtained from both sources, e.g. transitive property, symmetric property and equivalent (class/property). This clearly shows that the automatic transformation from database models to ontological models has some limitations, as there are many disparities between the two models.

Finally with all the advantages of a database, there are still some general limitations which might affect the extraction of semantics from database sources, such as the meaningless names of some tables or attributes, poor semantics and bad design intended to improve the performance. Our system tries to overcome these limitations.

5.8 The Transformation criteria

We differentiate the terms used to describe the process of converting database to ontology, such as mapping and transformation, and translating. The mapping process assumes the existence of both relational database and ontology. Here the approaches seek to obtain and set the corresponding variables between the two model structures, whereas transformation only assumes the existence of relational database. Therefore, the input of the transformation system is the source of the database and the result will be ontology; with the translating terms also being the assumption of the database existence. Where translating is referred to each aspect of database will receive a related corresponding construct without any modification to the source process, however, transformation usually involves some modification to the source before the correspondences matching the construct are found.

The criteria that should be considered by any approach, while transforming relational database into ontology are:

- **Preserve data**: The transformation should sufficiently depict the original data with its data types.
- **Preserve semantic**: only translate those constructs that have semantic equivalents.
- **Maintain the structure**: besides transforming tables and relationship, it should include constraints and a mechanism to migrate data.
- **Generality**: the system is not restricted to a specific domain.
- **Correctness**: the mechanism should prove the transformation accuracy. Mostly lack of correctness is caused by informal specifications in the transformation system.
- **Completeness**: the transformation rule should contain space in all possible cases of relations and relationship. Therefore, the transformation system needs a proof of completeness and the quality of the transformation should be evaluated.

The current approaches involve at least one or more of the following deficiencies.

- **Loss of inheritance**: The system should learn inheritance (hierarchy, multiple inheritances), because if the system fails in obtaining this characteristic then the ontology will have a flat structure.
- **Loss of constraints**: The system is unable to capture constraints on relationships.
- **Results are deduced based on erroneous database analysis**: Correct analysis for tables should include a multivalued table. Additionally, the correct analysis should differentiate between fragmentation case and hierarchy case.
- **Superfluous structure**: Additional tables representing fragment entities or multivalued table transformed to classes etc.
- **Theoretical transformation system**: Not implemented.
- **Domain expert dependent**: The system requires user intervention to decide between analysis options, which lead to a semi-automatic process while these

choices can be resolved and automated if the database were to experience a correct analysis.

- **Not considering the disparities between the two modelling:** Therefore, some constructs which shape the database structure can be translated into ontology constructs.
- **Not avoiding domain specific example influence**: This means approaches try to obtain some semantics from domain specific examples or from enumerating examples, then generalise their deductions which leads to incorrect rules. For example [46] use specific examples to represent symmetric and transitive properties, however their rules for generating property characteristics cannot be generalised.

5.9 The Transformation process

This section explains our proposed procedure to generate ontology from a relational database. This procedure consists of seven steps:

I. Using reverse engineering methodologies on a logical model written in SQL in order to capture syntax and schema structure (relations, attributes, inclusion dependences etc) and the hidden semantics.
II. Analyse the structures obtained and semantics in order to infer the best corresponding match constructed from the ontology side.
III. Apply the transformation rules to generate a complete ontology schema which includes classes, subsumption relationships, object properties and datatype properties.
IV. Checking the ontology's consistency through an ontology reasoner. In order to detect the superfluous relationships, and to remove redundant data.
V. Validate and refine the produced ontology by the information obtained from the conceptual database model.
VI. Evaluate the ontology.
VII. Migrate data in order to produce a knowledge base.

Our approach co-exists in semantic annotation in Semantic Web community and is known by the database community as reverse engineering [36].

5.10 Summary

This chapter showed the visionary framework (hybrid ontology approach) and how we utilise it in solving database integration problem. Then our approach architecture was demonstrated with the mechanism of creating each ontology elements and when and how each database source can be utilised. After that, the chapter mentioned the disparities between the database model and the ontology model in terms of aims and assumptions. Also it showed how that will affect the transformation system. Moreover, we compared between the semantics obtained by the EER model and The SQL-DDL source and the approach target ontology language (OWL) in order to specify the contexts that can be transformed and the ones that cannot. Finally, we addressed the criteria that ensure the success of the transformation system and our algorithm of applying them in our approach.

CHAPTER SIX: TRANSFORMATION FROM DATABASE TO ONTOLOGY

Objectives

- Specifying the assumptions.
- Explaining the formal rules from relational model to ontology model.

6.1 Introduction

This chapter will present the main rules for transforming the database to ontology. Section 2 gives an overview of the important features of our transformation rules. Sections 3 and 4 explain the transformation rules in detail and section 5 concludes the chapter.

6.2 Overview

Our system has many critical elements, which distinguish it from other works such as:

- Formal transformation rules eliminating syntactic and semantic ambiguities.
- The transformation rules in our system satisfy the completeness concept, since they cover all feasible relation cases encrypted in SQL-DDL, while considering the space of keys cases in terms of the interactions between the relations of primary key and foreign keys.
- Comprehensive rules, which contained all the possible situations in the database.
- Automatic approach with minimal requires user interactions.

Our system avoided the drawbacks of other approaches such as:

- The rules only being explained in the description in English.

- Expository rules without formality.
- Rules influenced by domain specific examples.

6.3 The rule source

We choose to establish our approach on the logical model of a database, instead of the conceptual model, since the database conceptual model (i.e. the ER model) is to a large extent not implemented during database design process, or the model had been lost. Another possible explanation may relate to undocumented changes altering the structure of the database. The logical data model has been written by SQL-DDL, which is used to represent physical schema part of relational databases for an enterprise in a standard way. In fact, SQL-DDL is not meant to be a knowledge representation language, although it is able to acquire some of the application domain semantics.

The assumption here is that our approach is capable of recovering the logical model of a database application, through analysing the syntactic elements of SQL-DDL which represent the physical code of the database model.

In their design, modern software programs have adopted the standard SQL language in order to express domain semantics in a rich way. However, SQL is not a representation form language, therefore inference procedures cannot utilise it. Thus the influence of our approach has been to convert SQL-DDL to the latest ontology language OWL which is already suitable to acknowledge a representation language.

6.4 Transformation Rules

This section has given formal rules for transformation of relational databases to OWL ontologies. First of all the predicates are defined; and subsequently an ontology has been constructed in three stages. The first stage, has built classes with their data type properties; whilst the second and third stages have successively explained how object properties are created and instances migrated respectively.

The transformation rules partition the database relations in such a way that no two rules can be applied to a single relation and no relation exists without a rule.

During the transformation process, we aimed to derive and preserve the semantics of database original model described by the definition of relations. Figure 6.1 shows the algorithm for applying the first stage of the transformation rules.

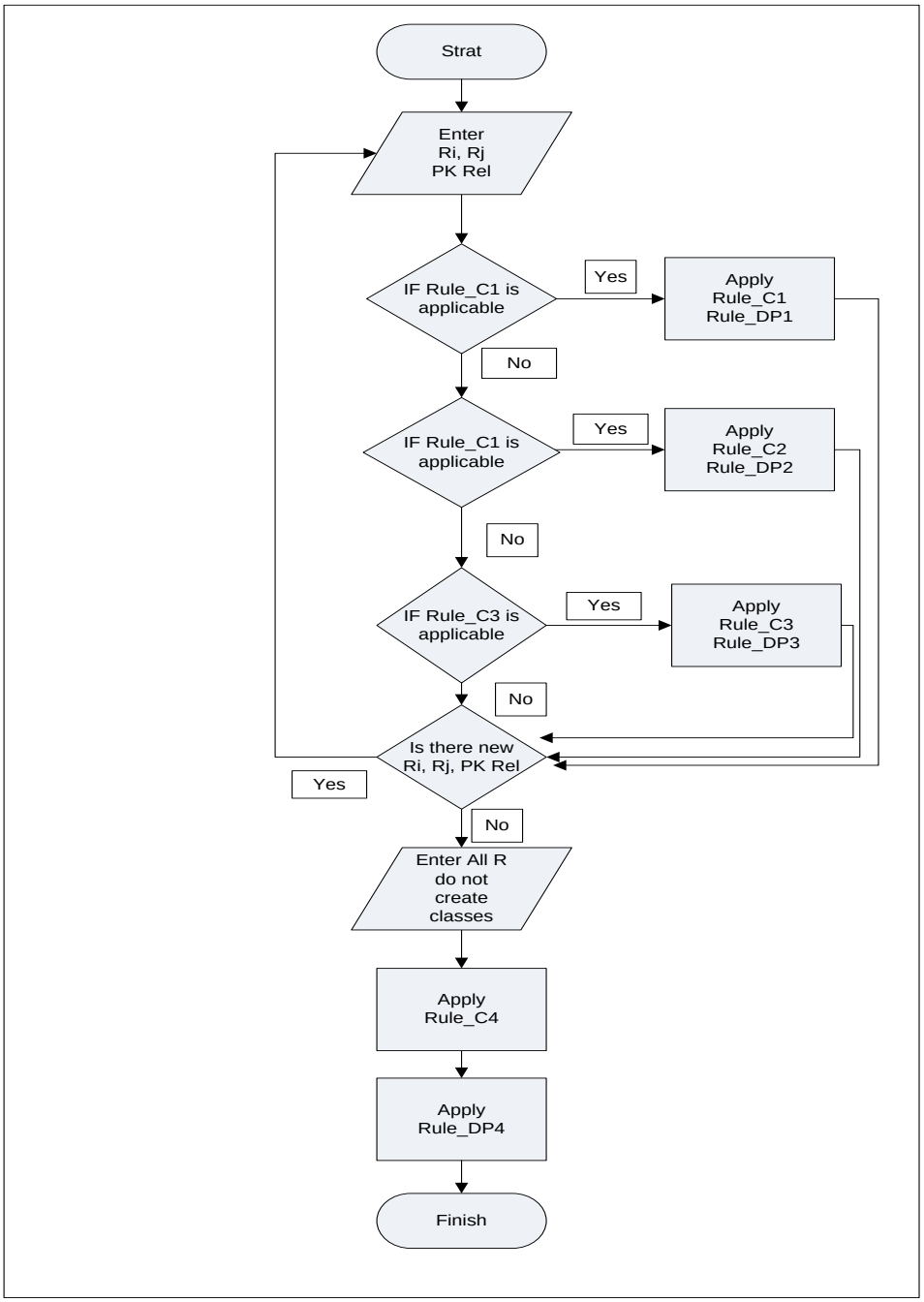


Figure 6-1: Classes and Data-type Rules Algorithm

6.4.1 Assumptions

In order to enhance our transformation rules from relational schema into ontology, we have adopted only the reasonable assumptions of other accomplished works, but have modified them to accommodate more database cases:

i. Relations are in the second normal form (2NF), available in SQL-DDL. Most extant approaches based their rules on 3NF database [36, 41, and 46]. See appendix A for more information about normalisation.
ii. New tuples added to the database are consistent with the derived metadata [43]. This condition only applied for bad designed databases.

The following explains the reasons behind the choice of these assumptions:

- The precise form which represents the current relational schema is available only in SQL DDL code.
- Any database necessitates changes in order to adapt the new requirements of the real world. The physical model should therefore be modified in order to express database evolvement. Also, as SQL DDL is the exclusive reflection of a database physical model, it should be the precise source representing database structure.
- Requiring the database to be in the third normal form is an impractical condition, since most available databases barely satisfy 2NF requirements and force database designers to apply normalisation algorithms [4] to capture functional dependencies. Then they must use them when changing the database structure to make the database qualify as the third normal form. Therefore we have modified our assumption to require databases to satisfy the second normal form only. The only exception for our approach is databases in the first normal form, which is not usually recommended by database designers since it has a lot of redundant data. However demoralised database, database in first normal form or second normal form would not lose any semantics. The real purpose of using demoralised database is to enhance the performance since it reduces the natural join between relations.

6.4.2 Predicates and Functions

We have enriched our transformation rules with some functions and predicates which clarify our system definitions.

Database predicates are defined in one or more arguments. Their purpose is to examine the relational database schema in order to capture a suitable construct that will satisfy the predicates' conditions. The list of predicates is shown below:

- *PK(x, R)*: *x* is the (single or composite) primary key of the relation *R* and has been represented as a set of attributes.
- *FK(x, R, y, S)*: *x* is a (single or composite) foreign key belonging to the *R* relation and refers to the primary key *y* of relation *S*.
- *Attr(x, R): x* is an attribute in relation *R*.

We have also defined the following functions:

- *OCC(v, x, R)*: there is a tuple in *R* for which the value of *x* is $v \in Type(x)$.
- *IsFK(x, R)* $\widehat{=} \exists y, S: FK(x, R, y, S)$.

  IsFK (x, R) means that *x* is a (single or composite) foreign key in the relation *R*.

- *NonFK(x, R): x is an* attribute belonging to the *R* relation which satisfies the condition of not participating in foreign keys.
- *NN(x,R) x* is an attribute part of the *R* relation constrained by NOT NULL; e.g. *NN(ID ,Employee)* holds.
- *UNQ(x,R) x* is an attribute belonging to the *R* relation constrained by the UNIQUE clause.

From another standpoint, ontology predicates are defined to represent the OWL ontology construct that has satisfied a set of conditions. These predicates are:

- *Class*(c) c is a class.
- *ObjP(p,d,r) p* is an object property with domain *d* and range *r* .
- $DP(x, c, type(x))$ *x* is a data type property with domain *c* and range *type(x).*

- *Inv(p,q)* when *p* and *q* are object properties, *p* is an inverse of *q.*
- *FP(p) p* is a functional property.
- *IFP(p) p* is an inverse functional property.
- *Crd(p,m,v)* the (max and min) cardinality of property *p* for class *m* is *v.*
- *MinC(p,m,v)* the min cardinality of property *p* for class *m* is *v.*
- *MaxC(p,m,v)* the max cardinality of property *p* for class *m* is *v.*
- *SubClass(m,n) m* is a subclass of class *n.*

We have also defined the following ontology functions:

- *Type(x)* maps an attribute *x* to its appropriate XML data type, as recommended.
- *one of(x)* is a function maps an attribute to the permitted values list; this function can be applied only when the constraint of CHECK IN is valid.
- ← Means the corresponding (relation or relationship or attribute).
- ≪ Means the class already exists from its corresponding relation.

However an *Inverse functional property (x) where x is a data type* is not available as it is part of OWL-Full. Tagging *Inverse Functional* to a data type property is permitted in OWL-Full only, which guarantees the uniqueness of an attribute. However, OWL-DL does not enable a datatype property to have an *Inverse Functional* clause, since the literals of OWL-DL are disjointed from *owl:Thing.*

6.4.3 Producing Unique Identifiers (URIs) and Labels

Producing unique identifiers for ontology constructs is essential in building ontology structures. OWL ontologies define their concepts, including classes and properties, by unique identifiers which are considered to be unique names. As in database each table and attribute has a unique identifier implied in its name. However database allows reusing attribute names as long as they belong to different tables.

Since the relational schema labels its concepts; therefore it is possible to use them in the ontology with an exception of no duplication is allowable.

There are two ways to resolve the duplication problem. The first idea has been to append the name borrowed from relational schema with random numbers to make it unique. The second solution to produce URIs based on fully qualified names of schema elements. User intervention is needed to ensure second solution success. Here a domain expert is preferable to specify qualified names to ontology constructs. We preferred the second solution since most database designers abridge names of relations and attributes in database relational schema. Consequently, these abridged names might have a meaning indication and sometimes it is difficult to infer the meaning. The second reason behind this choice was to make the ontology human readable and prepare it for the ontology alignment step.

6.4.4 Class and Data type property Creation Rules

Our transformation rules have the form: IF condition THEN action. In the following rules, relation symbols are assumed to be universally quantified. The rules consider maintaining all predicts of relational database in the conditional side; and all ontological predicts in the action side. This characteristic has distinguished our rules from others [45]; whereby their database predicates appeared in both the head and body of a rule.

6.4.4.1 Fragmentation rules

In this rule we discussed both class and data type creation for this case.

- **Rule C_1: Fragmentation class rule**

If the information of one entity type has spread across more than one relation, known as fragmentation (vertical partitioning of a table), then it should be integrated into one class.

There are three ways to deal with fragmentation tables. Firstly, if the database has been correctly and completely specified; then the fragmentation could be distinguishable from inheritance, as Rule-C_{1cc} showed the specification. Whereas, if the database has not been completely specified, but has complete and available data, then the number of records in the master and the slave tables must be identical to consider the relationship

as fragmentation. The third way has been to treat fragmentation as inheritance as far as database in the third normal form as long; and as the relationship from type One - To - One.

Details on how to choose the better way can be gleaned in the discussion section.

$$Rule\_C_{1cc}\begin{cases} -\ \text{Class condition:} \\ \qquad \exists x, y : \begin{pmatrix} (a)PK(x, R_1)\ \wedge PK(y, R_2)\ \wedge \\ (b)FK(y, R_2, x, R_1) \wedge \\ (c)FK(x, R_1, y, R_2) \end{pmatrix} \\ -\ \text{Class Action:} \\ \qquad class\ (C)\ \leftarrow\ (R_1 \cup R_2) \end{cases} \tag{1}$$

The need for fragmentation of an entity to more than one table arises when one entity has many attributes. Some of these attributes are mainly optional; not applicable for all tuples and usually for the majority of time assigned a NULL value. Therefore the database designer can split the information into more than one table. The important fields are then assigned to the main table which will be accessed more often; and the insignificant fields can be stored in another table(s). The benefit associated from this procedure can be improved performance.

Another need for fragmentation has been when one entity has two parts of information. The classified (confidential) fields need to be stored within high level security; and can only be accessed or manipulated by the administrator. The other part can be the public fields accessible by all users. The third need for the fragmentation technique is for the distributed database. When part of one entity has been manipulated in the branches of the database; then the update could take place at a later stage in the main database. For example the employee entity may have all the following attributes: employee SSN, employee-id-number, first-name, family-name, DOB, address, speciality, extension number, home phone number, mobile phone number, fax number, employee website and email etc.

These attributes or important information could be assigned to one table such as the employee table; and the rest could be assigned to an employee - details table.

- **Rule DP_1: Fragmentation data type property rule**

All present attributes in each relation participating in the fragmentation rule, and not forming a foreign key, will be allocated to the integrated class created from Rule C_{1cc}, as depicted in equation 2. For example, if the entity employee has two tables. The first table - Employee - have the attributes: (**SSN**, employee-id, first-name, family-name, DOB, speciality); all of which do not form foreign keys except the (**SSN**) attribute. Similarly, the second table - Employee Details - have the attributes: (**SSN**, address, speciality, extension number, home phone number, mobile phone number, fax number, employee homepage and email) of which all the attributes do not form foreign keys, except the (**SSN**) attribute in the - Employee Details - table**.** The repeated attributes appear once in the integrated class which is the primary key of the two tables. A data type property can then be created for each attribute in these two tables and allocated to the one class – Employee - which correspond to both - Employee - and - Employee Details - tables. These data type properties will thus consider the class - employee- as their domain, and the corresponding data type in Table 6.1 as their range. We have referred to them by *type(x)* in our rules. For example, *first-name* will have class - employee - as its domain and the range of *xsd*: *string*.

$$Rule_{DP_{1cc}} \begin{cases} - \text{ Datatype condition:} \\ \quad \begin{pmatrix} Attr(x, R_1) \wedge NonFK\ (x, R_1) \vee Attr(PK(R_1)) \\ \vee (Attr(y, R_2) \wedge NonFK\ (y, R_2)) \end{pmatrix} \\ - \text{ Datatype action: } create \\ \quad DP(x, c, type(x)) \end{cases} \qquad (2)$$

Table 6-1: Data type between the Database and XML

Type	DB	XML Schema data type
number	smallint	Xsd:short
	integer/int	Xsd: integer
	float	Xsd:float
	real	
	decimal	Xsd:decimal
	numeric	
	Double precision	Xsd:double
Char	char /varchar/vchar	Xsd:string
date/time	time	Xsd:time

Type	DB	XML Schema data type
	date	Xsd:date
	Date/time(TIMESTAMP)	Xsd:datetime
	interval	duration
Boolean	Bit/Boolean	Xsd:Boolean
**	Bit varying	byte

6.4.4.2 Hierarchy rules

This rule has represented the case of inheritance for both class creation and its data types.

- **Rule C_2: Hierarchy class rule**

One of the most important stages in building the ontology is the construction of the hierarchy. The term 'hierarchy' referred to the specification of the relationships between classes and subclasses. The class-subclass relationship appeared as an IS-A relationship between entities in the EER model. Rule- C_2 demonstrated the conditions and the action for this rule.

$$Rule\_C_2 \begin{cases} -\ \text{Class condition:} \\ \quad \exists x, y : \begin{pmatrix} (a) PK(x, R_1) \ \wedge PK(y, R_2) \ \wedge \\ (b) FK(y, R_2, x, R_1) \end{pmatrix} \\ -\ \text{Class action: } create \\ \qquad\qquad \begin{pmatrix} 1)\ \ class\ (C_1) \leftarrow\ R_1 \\ 2)\ class\ (C_2)\ \leftarrow\ R_2 \\ 3)\ \ SubClass\ (C_2, C_1) \end{pmatrix} \\ \qquad where \\ \qquad\qquad C_1\ and\ C_2\ are\ not\ already\ created \end{cases} \tag{3}$$

Some approaches [41] [42] failed to distinguish between the fragmentation of one entity into more than one relation; and the IS-A relationship, which represented two different entities, with the idea of super-entity and sub-entity (generalisation /specialisation). Other approaches have enabled the user to decide [36]. This failure came from incorrect specifications of the fragmentation case and therefore the fragmentation and hierarchy case have the same representation.

However our approach distinguished between the two cases by the foreign key restriction. Both fragmentation and hierarchy had the same first condition of tables sharing the same primary key. The fragmentation case required the primary keys to act as foreign keys referring to each other; whereas the hierarchy case required only one of the primary keys to act as a foreign key to others. Even if there was a chain of class and subclass hierarchies for more than two levels; the condition remained true for inheritance, since the two tables do not refer to each other. There was noticeably no restriction on the master relation primary key, whether it was a foreign key or not, which allowed the master relation to be subclass from a superclass. This meant the rule was applicable even if the master relation preceded its primary key as foreign key or not. This provided this rule power to build many level inheritance hierarchies.

Suppose we had three tables T1, T2 and T3. The primary key of these three tables are identical; whilst at the same time the T2 primary key referred to T1 and T3 primary key referred to T2. When the rule is applied between T2 and T3 the result is class for T2 and subclass for T3. Subsequent applying the same rule between T1 and T2; T1 would be superclass and T2 as subclass. The third probability with regards to the relationship between T1 and T3 was not applicable.

In addition our approach, through good observation, inferred an implicit IS-A relationship even when the foreign key was disjointed from the primary key; for instance the two tables below depicted by Person and Student. Here a foreign key Student table has two restrictions of UNIQUE and NOT NULL and whenever these two characteristics combined in a foreign key it could form an IS-A relationship.

Person (P-ID (Primary Key), DOB, Name...)

Graduate Student (St-ID (Primary Key), Research- topic, ID UNIQUE NOT NULL

FOREIGN KEY REFERENCES Person (P-ID))

Generally placement of a foreign key into a table has been the standard way to model a relationship for (One-To-Many), and (One-To-One) relationships including the IS-A relationship.

However Tirmizi *et al* [45] argued that if the IS-A relationship was modelled; but the primary key of the superclass did not become the primary key of the subclass, then, there could be no assumption of the existence of an IS-A relationship. However from the example of Student and Person tables, Graduate Student IS-A Person. Nonetheless because the primary key of the Person table (**P-ID**) did not become the primary key of the Graduate Student table, we cannot assume the existence of an IS-A relationship.

Conversely (**P-ID**) was placed in the Graduate Student table as a foreign key referencing Person table, at the same time, declared to be UNIQUE and NOT NULL. This effectively declared that (**P-ID**) was an alternate key and thus, should declare an ISA relationship.

The Rule_C_{2nu} thus represented the NOT NULL and UNIQUE relationship as outlined:

$$Rule\_C_{2nu}\begin{cases} -\ \text{Class condition:} \\ \quad \exists x, y : \begin{pmatrix} (a) PK(x, R_1)\ \wedge \neg PK(y, R_2)\ \wedge \\ (b) FK(y, R_2, x, R_1) \wedge \\ (c) NN(y, R_2)\ \ \wedge \\ (d) UNQ(y, R_2) \end{pmatrix} \\ -\ \text{Class Action:} \\ \quad \begin{pmatrix} 1)\ \ class\ (C_1) \leftarrow\ R_1 \\ 2)\ \ class\ (C_2)\ \leftarrow\ R_2 \\ 3)\ \ subClass\ (C_2, C_1) \end{pmatrix} \end{cases} \tag{4}$$

Using this case has been rare; however to model multiple inheritance this can be considered the only solution. If we define the table of Student and redefine Graduate Student table as:

Student (St-ID (Primary Key), Major, Degree)
Graduate Student (St-ID (Primary Key), Research- topic,
ID unique not null FOREIGN KEY REFERENCES Person (P-ID),
St-ID FOREIGN KEY REFERENCES Student (St_ID))

The class of Graduate Student will thus inherit all the properties of class Person and class Student.

i. Rule DP_2: Hierarchy data type property rule

The order of building the hierarchy chain was therefore very important for this step. For example if we had three relations: T1 super-relation; T2 sub-relation; and T3 sub of the sub-relation then the first relationship to deal with would be, between T1 and T2, since T1 does not refer to other relation. The approach thenceforth deals with the relationship between T2 and T3.

The reason behind this procedure was to ensure there was no repetition attributes assigned to both the super-class and sub-class. Here the difference between database design and ontology design, in maintaining the inheritance, was the database need to repeat the primary key throughout relations. The primary key performs as foreign key referring to its super relation in order to represent generalisation; whereas ontology only used the term of subclass.

Rule DP2 demonstrated the data type property rule for relations, participating in hierarchy rules. From the class hierarchy Rule C2, we created two classes, one for the master relation and the other for the sub-relation. This ensured that all the attributes present in each relation would be moved to the corresponding classes without repetition.

$$Rule_{DP2}\begin{cases} -\ \text{Datatype condition:} \\ \qquad \begin{pmatrix} Attr(x,R_1) \wedge NonFK\ (x,R_1) \checkmark \\ Attr(y,R_2) \wedge NonFK\ (y,R_2) \end{pmatrix} \\ -\ \text{Datatype action}: create \\ \qquad \begin{pmatrix} 1)\ DP\big(x,C_1,type(x)\big) \\ 2)\ DP\big(y,C_2,type(y)\big) \end{pmatrix} \end{cases} \tag{5}$$

- **Discussion:**

Both fragmentation and hierarchy shared the same specification in many incorrect relational schemas and a relationship of type One-To-One. This type of relationship was mandatory from both sides, in which the EER diagram has shown explicitly the hierarchy situation. However, after converting the EER of the database to relational schema; the hierarchy relationship became ambiguous. For example a table can be a result of an entity or a relationship; and one entity can be articulated in more than one table.

There are therefore three solutions to determine whether the One-To-One relationship belong to the fragmentation or hierarchy case.

The first solution considers fragmentation as a type of hierarchy. It is therefore acceptable to treat the vertically partitioned table as inheritance as the assumption of database is in the third normal form (3NF) and could be applied by Rule-C_2. This is true from a database perspective; but not acceptable from an ontology perspective.

The approach [45] claimed that partition of one entity into more than one table would violate the third normal form. However there were no obvious functional dependencies that would violate 3NF for decomposing a table. Violations of 3NF, not violations of 1NF, would not alter the primary keys in any way upon decomposition. However, a violation of 1NF would technically be a violation of 3NF; and could cause a designing problem. The following example shows how normalisation can affect the database design:

Professor-Teaching (Prof-ID (Primary Key), department, <List of Courses Taught>) Professor-Personal-Details (Prof-ID (Primary Key), DOB, Spouse-Name, FOREIGN KEY Prof-ID REFERENCES Professor-Teaching)

In the above example the two tables meet the requirement of hierarchy; however clearly, the Professor-Personal-Details table is not a subclass of the Professor-Teaching table. Furthermore, if the Professor-Teaching table was properly normalized, (**Prof-ID**) would not be its primary key and thus (**Prof-ID**) could not be a foreign key in Professor-Personal-Details table.

Other approaches have claimed that there is no way to distinguish between tables produced from entities with an ISA relationship; and one entity representing vertical partition. We would argue that, if the database has been **correctly** and **completely** specified, their claim is invalid. For example the two tables below:

Professor-Personal-Details (Prof-ID (Primary Key), DOB, SpouseName)

Professor-Teaching (Prof-ID (Primary Key), department, NumberOfCoursesTaught , FOREIGN KEY Prof-ID REFERENCES Professor-Personal-Details).

Both tables share the same primary key. The second table primary key is a foreign key referring to the first table. The argument is that we cannot specify whether this relationship represents an ISA or a vertical Partition. However, if the database has been well designed and completely specified; the Professor-Personal-Details table would also need to specify (**Prof-ID**) as a foreign key, referring to Professor-Teaching table to represent vertical partition. Thus, the above form should only be interpreted as an ISA relationship. The vertically partitioned version should be specified as:

Professor-Personal-Details (Prof-ID (Primary Key), DOB, SpouseName,

FOREIGN KEY Prof-ID REFERENCES Professor-Teaching)

Professor-Teaching (Prof-ID (Primary Key), department, NumberOfCoursesTaught, FOREIGN KEY Prof-ID REFERENCES Professor-Personal-Details)

Thus, it was possible to distinguish the two cases in a completely specified database. The second solution of tables representing fragmentation must identify the shared primary key as a foreign key in all tables. Unfortunately, this solution has not usually been utilised in practice due to the fact that we would have needed to insert a record into two tables simultaneously in order to preserve referential integrity.

The third solution related to counting table records, if the database had complete and available data. The fragmentation rule would be rewritten for record counting as:

$$Rule\_C_{1count} \begin{cases} - \text{ Class condition} \\ \quad \exists x_1, x_2, \dots, x_n : \\ \quad \begin{pmatrix} (a) \wedge_{i=1}^{n} PK(x_i, R_i) \wedge \neg isFK(x_1, R_1) \wedge \\ (b) \wedge_{i=2}^{n} FK(x_i, R_i, x_1, R_1) \wedge \\ (c) \forall v \in type(x_1): (OCC(v, x_1, R_1) \Rightarrow \wedge_{i=2}^{n} OCC(v, x_i, R_i)) \end{pmatrix} \\ \\ - \text{ Class action: } create \\ \qquad\qquad class\ (C) \leftarrow \left(\bigcup_{i=1}^{n} R_i \right) \end{cases} \quad (6)$$

To apply this rule, all participating relations must satisfy all the conditions to be merged into one class.

i. There are two or more relations sharing the same primary key.
ii. There is a master relation whereby:
    a. its primary key is not a foreign key.
    b. the primary keys of all the remaining relations also act as foreign keys referring to the master relation.
iii. All the primary key values present in the master relation must exist in all remaining relations.

If all these three conditions are satisfied then we can merge these tables into one class in the ontology.

This relationship was of the One-To-One cardinality ratio and total participation constraint. In (min, max) notation is (1, 1) on each side. The rule for data type needed to be altered to:

$$Rule_{DP\,1count}\begin{cases} -\ \text{Datatype condition:} \\ \quad \bigwedge_{i=1}^{n}(Attr(x,R_i)\ \wedge NonFK(x,R_i)) \\ -\ \text{Datatype action: } create \\ \quad DP(x,c,type(x)) \end{cases} \tag{7}$$

To extract the inheritance relationship, we required the analysis of tuples in each relation to satisfy the conditions.

The condition and the action illustrate the rules for inheritance:

$$Rule\_C_{2\ count}\begin{cases} -\ \text{Class condition:} \\ \quad \exists x,y: \begin{pmatrix} (a)PK(x,R_1)\ \wedge PK(y,R_2)\ \wedge \\ (b)FK(y,R_2,x,R_1)\wedge \\ (c)\exists\ v\ \in type(x): (OCC(v,x,R_1)\wedge \neg\ OCC(v,y,R_2)) \end{pmatrix} \\ -\ \text{Class action: } create \\ \quad \begin{pmatrix} 1)\ \ class\ (C_1) \leftarrow\ R_1 \\ 2)\ \ class\ (C_2)\ \leftarrow\ R_2 \\ 3)\ \ SubClass\ (C_2,C_1) \end{pmatrix} \\ \quad where \\ \quad C_1\ and\ C_2\ are\ not\ already\ created \end{cases} \tag{8}$$

To apply this rule, all participating relations must satisfy all the conditions to create two classes (superclass, subclass).

1. There are two relations sharing the same primary key.
2. There is a master relation where
    a. Its primary key is not a foreign key or one referencing third relation; for the former case it became a master relation. Otherwise there would be hierarchy inheritances. The hierarchy inheritances are allowable in our rule since there are no conditions on the master relation primary key preventing it from referencing a third relation.

   b. The primary key of the other relation also works as a foreign key (referring to the sub-master or to the master relation).

3. Not all the primary key values present in the master relation must exist in the other relation.

When this rule was applied, we could build a superclass from the master table and the other table would be a subclass in the ontology.

We considered this relationship to have one-to-one cardinality. The participation ratio must be (1, 1) in the master relation and (0, 1) in the other relation. There was no need for any modification to the data type rule for the hierarchy case.

This solution has been based on table record counting. From rules $Rule\_C_{1\ count}$ and $Rule\_C_{2\ count}$ the preconditions (a) and (b) are the same for two relations, so the key solution is in the precondition (c).

At the implementation level, our approach used a number of tuples to represent the precondition (c), without need for user participation.

We considered the number of tuples to check for both the fragmentation of an entity or the IS-A relationship. Thus, record counting could be facilitated by using the SQL query statement to represent the precondition (c) in $Rule\_C_{1\ count}$ and $Rule\_C_{2\ count}$ as below:

NT = Select count (*) from table name;

Function (NT) will therefore return the number of tuples of the relation. We used the comparison with the number of tuples:

If rule (a, b) in Rule C_{1count} satisfied and NT (R1) = NT (R2).

The system subsequently integrated all relations into one class. For each record in the first relation there existed a record corresponding to it in the other relations, whereas:

If rule (a, b) in Rule C_{1count} satisfied and NT (R1) > NT (R2),

the system successively built two different classes and created a subclass relationship between them.

For example, the tables of Professor and Professor -Photo have the same primary key and represent one entity.

```
Professor (Prof-ID (Primary Key), DOB, Name, Address...)
Professor-Photo (Prof-ID (Primary Key), DigitalPhoto,

FOREIGN KEY Prof-ID REFERENCES Professor)
```

Sharing the same primary key between the two tables, and that of the Professor-Photo table refers to the primary key of the Professor table, thus representative of the One-To-One relationship.

For each tuple in the Professor table, there must be a tuple in the Professor-Photo table:

*NT (*Professor*) =NT (*Professor -Photo*),*

The equality of fragment tables can be handled by assuming the data entry required for each new record in the Professor table and that there will be an entry of (ID, NULL) to Professor-Photo table if no picture yet exists. Therefore for each new primary key value, inserted in the master table, there would be an insertion for the primary key value in the slave table simultaneously. This could be easily completed by a trigger as shown:

```
CREATE OR REPLACE TRIGGER Audit_Prof
  AFTER INSERT ON Professor
    FOR EACH ROW
BEGIN
     insert into professor –Photo (Prof-id) values (:new. Prof-id);

      insert into professor –Photo (DigitalPhoto) values(Null);
END;
```

The count option could not be utilised unless there was a condition allowing the insertion in both fragment tables, as seen by the trigger above. Conversely, if there were tables for Student and Graduate-Student represented by two different entities from the specification below, the two tables both satisfy rule (a, b) in Rule C_{1count}.

Student (St-ID (Primary Key), DOB, Name, Major...)

Graduate-Student (St-ID (Primary Key), Research-Area, FOREIGN KEY St-ID

REFERENCES Student)

At this point *NT* (Student) > *NT* (Graduate-Student) is therefore true,

This was obvious because the Student table had the Bachelor, Master and PhD student records; whilst the Graduate-Student table only had the Master and PhD students. However, if someone claimed that, the number of tuples in the Student table was equal to the number of tuples in the Graduate-student, then all the preconditions in Rule-C_{1count} could be satisfied, and the two tables would merge into one class. From this we decided that this university only had graduate students, so there was no need to build two classes for student entity.

The completeness and availability of data was our only concern in using the row counts to distinguish between the IS-A relationship and vertical partitions. Incomplete data meant there were records related to other records not yet added to the database. For instance in the example of Professor and Professor-Photo tables the row count of the two tables may have been an exact match in the creation time. However, it would be possible to have a record for Professor without yet entering the picture. If the picture had not yet been added, then the counts and results would be different; unless the approach activated dealing with triggers. Conversely, the counts could be different when the database was first created, but updated shortly thereafter.

From Student and Graduate-Student, examples the row counts should be different if there were undergraduate students. The concern here would be if there were undergraduate students; but their information had not yet been entered.

Therefore there might be two slightly different cases:

i. There are master records that have no relationship with the slave records not yet entered. Therefore the numbers of records match; when they should not.
ii. There are slave records that have not yet been entered, which should have been; therefore the numbers of records do not match when they should.

We overcame the second problem by requiring the existence of a simultaneous insertion trigger. For both cases we rejected the incomplete database by the assumption of the derived metadata are consistent and would not be affected by any new tuple added to the database.

6.4.4.3 Multi-valued rules

Most approaches neglected the fact of the existence of the relation corresponding to multivalued attribute; only the EER based approach took care of this case. However all the SQL and database schema approaches did not mention this case.

- **Rule C_3: Multi-valued class rule**

Databases cannot deal directly with a multi-valued attribute; whilst ontologies have an efficient way to directly deal with this. For example, the table of Student has the attribute Hobbies as in Fig. 6.2. Here the database designer had two choices to represent this attribute. The first choice was to put all the hobbies for each student into one field, separated by commas (see Figure. 6.2A). The Database designer considered this choice as bad design since it violated the first normal form, adopting instead the second choice, which put the multi-valued attribute in separate relation to avoid duplication of tuples. The designer then linked the multi-valued relation, with the master, by placing the primary key of the master relation as part of the primary key of the multi-valued attribute relation as in Figure. 6.2B.

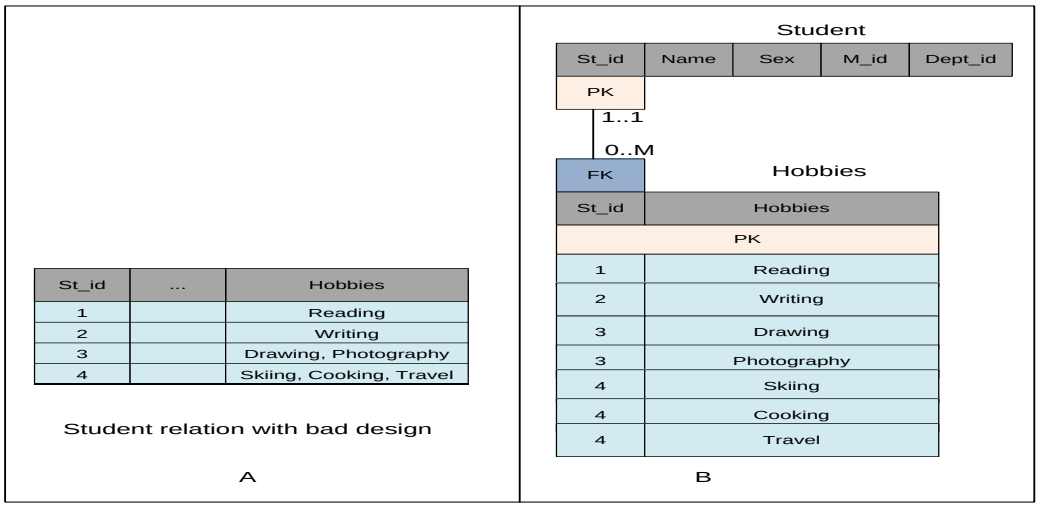


Figure 6-2: Representation of Multi-Valued Attributes

From our example, the attributes (**st_id, hobby**) form the primary key of the hobbies relation. Hence, the multi-valued attribute relation appeared as a weak entity. However, the difference between the weak entity and the multi-valued attribute relation was simple; because the multi-valued attribute relation had just one attribute besides the primary key of the master relation. Conversely, the weak relation had more than one attribute to describe the relation, as well as the primary key of the master relation.

Therefore, in order to represent an entity with a multi-valued attribute in ontology, we merged the master relation and the multi-valued relation into one class. This was facilitated by the ability of OWL to combine a single valued attribute with a multi-valued attribute in the same instance. Rule C_3 has clarified this idea:

$$Rule\_C_3 \begin{cases} -\ \text{Class condition:} \\ \quad \exists x, y : \begin{pmatrix} (a)\ PK(x \cup y, R_2)\ \wedge \\ (b)\ \exists z : FK(x, R_2, z, R_1) \wedge \\ (c)\ Attr(y, R_2)\ \wedge NonFK(y, R_2) \end{pmatrix} \\ -\text{Class Action: } create \\ \qquad\qquad class\ (C)\ \leftarrow\ (R_1 \cup R_2) \end{cases} \qquad (9)$$

The difference between the multi-valued case and that of the fragmentation rule was the type of relationship involved. In the fragmentation case, the relationship was of the One-To-One type; whilst in the multi-valued attribute, the relationship would be of the One-To-Many type.

- **Rule DP_3: Multi-valued data type property rule**

The multi-valued relation had two attributes, one holding the relationship with the master relation and the other representing the multi-valued attribute. Here we integrated the multi-value attribute into the class which corresponded to the master relation. In our example, the relation Hobbies represented the multi-valued attribute which contained the attributes (st_id, hobby_name); whereby the relation Hobbies branched from the relation Student. Thus, the attribute hobby_name would be assigned to the class Student and considered as its domain. In addition the repeated attribute only appeared once in the main class. In our example the st_id would not be repeated in Student class. The data type property representing the multivalued attribute must not be functional and does not have cardinality restriction.

$$Rule_{DP\,3}\begin{cases} -\ \text{Datatype condition:} \\ \qquad a.\,Attr(x,R_1)\wedge NonFK(x,R_1) \\ \qquad b.\,Attr(y,R_2)\wedge NonFK(y,R_2) \\ -\ \text{Datatype action: } create \\ \qquad DP\big(x,C,type(x)\big) \\ \qquad DP\big(y,C,type(y)\big)\ \wedge\ \neg FP(x) \end{cases} \tag{10}$$

6.4.4.4 Default rules

All tables which do not satisfy the previous rules will automatically direct to the default rule.

- **Rule C_4: Default class rule**

The condition and the action below demonstrate the default rule. Where rules C_1, C_2 and C_3 are inapplicable, then all remaining relations, not representing a binary relationship

with many-to-many cardinality, form a class. The default rule will form classes for strong and weak entities. Strong entities forming classes are obvious; however weak entities can be treated like strong entities, since they may have relationships to entities other than their corresponding strong entity.

$$Rule\_c_4 \begin{cases} -\ \text{Class condition:} \\ \quad \neg\, \exists x, y : \begin{pmatrix} (a)\ PK(x \cup y, R)\ \wedge\ isFK(x, R)\ \wedge\ isFK(y, R)\ \wedge \\ (b)\ \text{x} \cap \text{y} = \emptyset\ \wedge \forall z : (\ isFK(z, R) \Rightarrow z\ \in \{x, y\}\ ) \wedge \\ (c)\ \ \forall \text{t} : (Attr(t, R) \Rightarrow t\ \in\ x \cup y) \end{pmatrix} \\ -\ \text{Class action:}\quad create \\ \qquad\qquad\qquad\qquad class\ (C)\ \leftarrow R \end{cases} \tag{11}$$

Besides the cases of strong and weak entities, this rule was applicable for the two following cases as well.

- Case 1: Binary relationships with additional attributes

Most approaches do not consider the existence of additional attributes describing a Many-To-Many relationship, for example the Result relation having a Many-To-Many cardinality ratio with the grade attribute (see Table 6.2 for the schema).

Table 6-2: Schema for Attributes on relationship

Relation	**Primary Key(s)**	**Foreign Key(s)**
Result (st_id, c_id, grade)	st_id, c_id,	st_id (Student) c_id (Course)

Here we proposed to create a new class for this kind of relationship with two pairs of inverse object properties; and to create a data type property for the additional attribute (see Fig. 6.3). The creation of object properties has been discussed in detail in subsection (6.4.5).

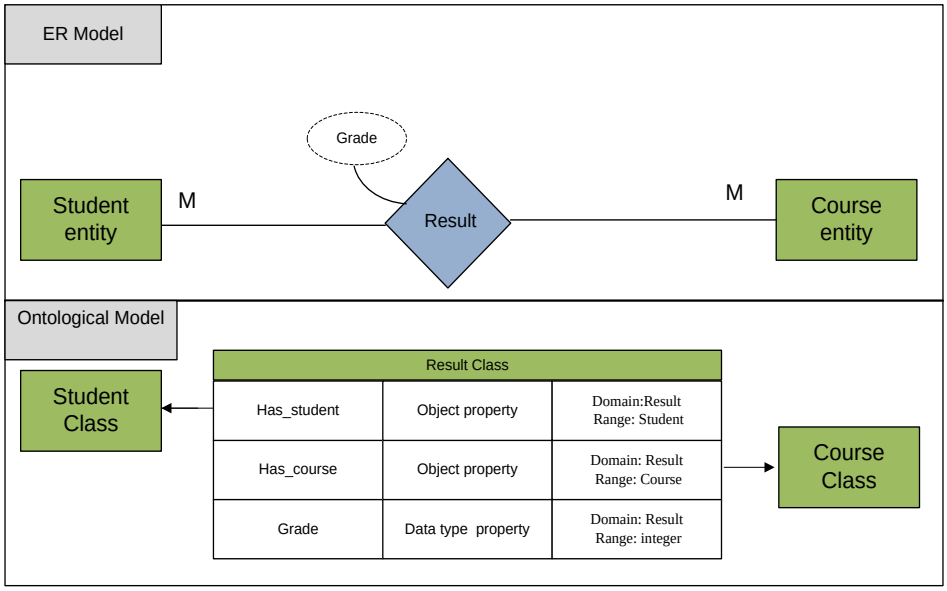


Figure 6-3: Many-to-many relationships with additional attributes

Noticeably the approach dealt with the number of referenced relations instead of the number of attributes in the foreign key to avoid the composite attribute. This would remove any confusion caused by a foreign key formed by more than one attribute.

[36] [41] failed to provide a general rule to decide whether a relationship was of the type Many-To- Many or not; because their rules supposed that each relation had just one attribute for the primary key. They did not consider the Many-To-Many relationship between a strong entity whose primary key might have one attribute and a weak entity with a primary key having more than one attribute.

- Case 2: N-ary relationships

OWL did not deal with N-ary relationships when N>2, so some approaches such as [41] [42] suggested decomposing any ternary or N-ary relationship to a binary relationship. Our approach first created a class to deal with ternary or higher relationships, and then decomposed the ternary or the N-ary relationship to binary relationships with cardinality set to one [50]. This was the only way to ensure that such a relationship existed as a whole.

An example will thus explain the necessity of creating a class for an N-ary relationship. Suppose we have the ternary relationship Teach which includes the Student and the Course and the Staff as in Table 6.3.

Table 6-3: Schema for ternary relationship

Relation	**Primary Key(s)**	**Foreign Key(s)**
Teach (st_id, c_id, Staff_id)	st_id, c_id, Staff_id	st_id (Student) c_id (Course) Staff_id (Staff)

When we decompose this to binary relationships, we would get three relations (see Fig. 6.4). The figure illustrates the repetition of data in all three tables, which violate the characteristic of the relational model by duplicating tuples, thus affecting the design of the ontology as well. Therefore each N-ary relationship will be represented by a class and two object properties with cardinality set to one for each binary relationship part of N-ary decomposition.

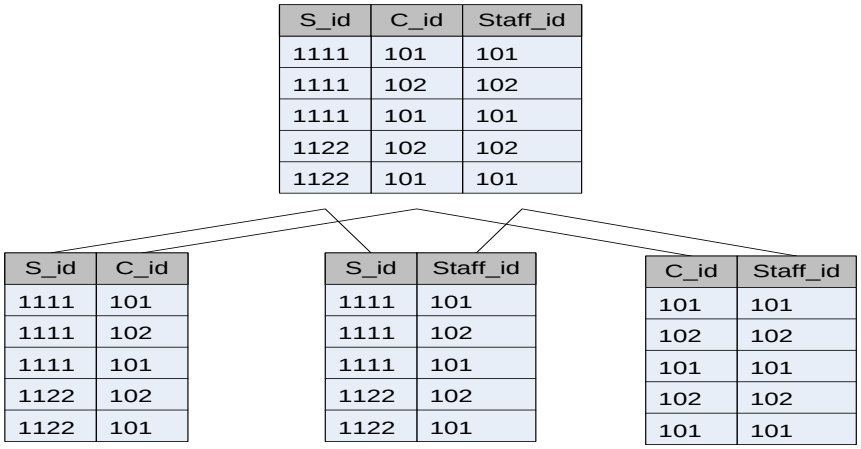


Figure 6-4: Ternary relation after decomposing

- **Rule DP_4: Default data type property rule**

For all relations, not participating in any of the rules DP_1, DP_2 or DP_3, data type properties could be created from their attributes which did not form foreign keys as well. Each data type property would be allocated to its corresponding class.

$$Rule\_DP_4 \begin{cases} - \text{ Datatype condition:} \\ \qquad Attr(x,R) \wedge NonFK(x,R) \\ - \text{ Datatype action}: \ create \\ \qquad DP(x, C, type(x)) \end{cases} \tag{12}$$

We did not use a general rule for creating data type property for all cases as in [41, 46]; instead we made specific data type property rules for each class rule. This insured each class had its own data type property; and thus preserved the multivalued and fragmentation table attributes.

- **Applying other SQL aspects to data type property rules**

Specifying the SQL constraints was possible (like DEFAULT, NOT NULL, UNIQUE etc.) to OWL language. In fact, our approach attempted to map database constraints into OWL axioms, while preserving the semantics of the original database schema.

- ***General Data type axiom algorithm**:(13)*

All attributes below must first satisfy the two conditions:

i. Not a foreign key
ii. Not a multivalued attribute

We could then apply the SQL aspects as outlined:

I. If the attribute *x* is NOT NULL then gloss the attribute *x* with *MinC =1*.
II. If the attribute *x* is UNIQUE then make it *MaxC =1*.
III. If the attribute *x* is a primary key or (NOT NULL and UNIQUE) then make it *Crd*=1.
IV. If the attribute *x* is DEFAULT then make it *Functional* and *hasValue*.

V. If the attribute *x* is CHECK IN then make it *oneOf*.

VI. Otherwise any attribute annotate with *Functional*.

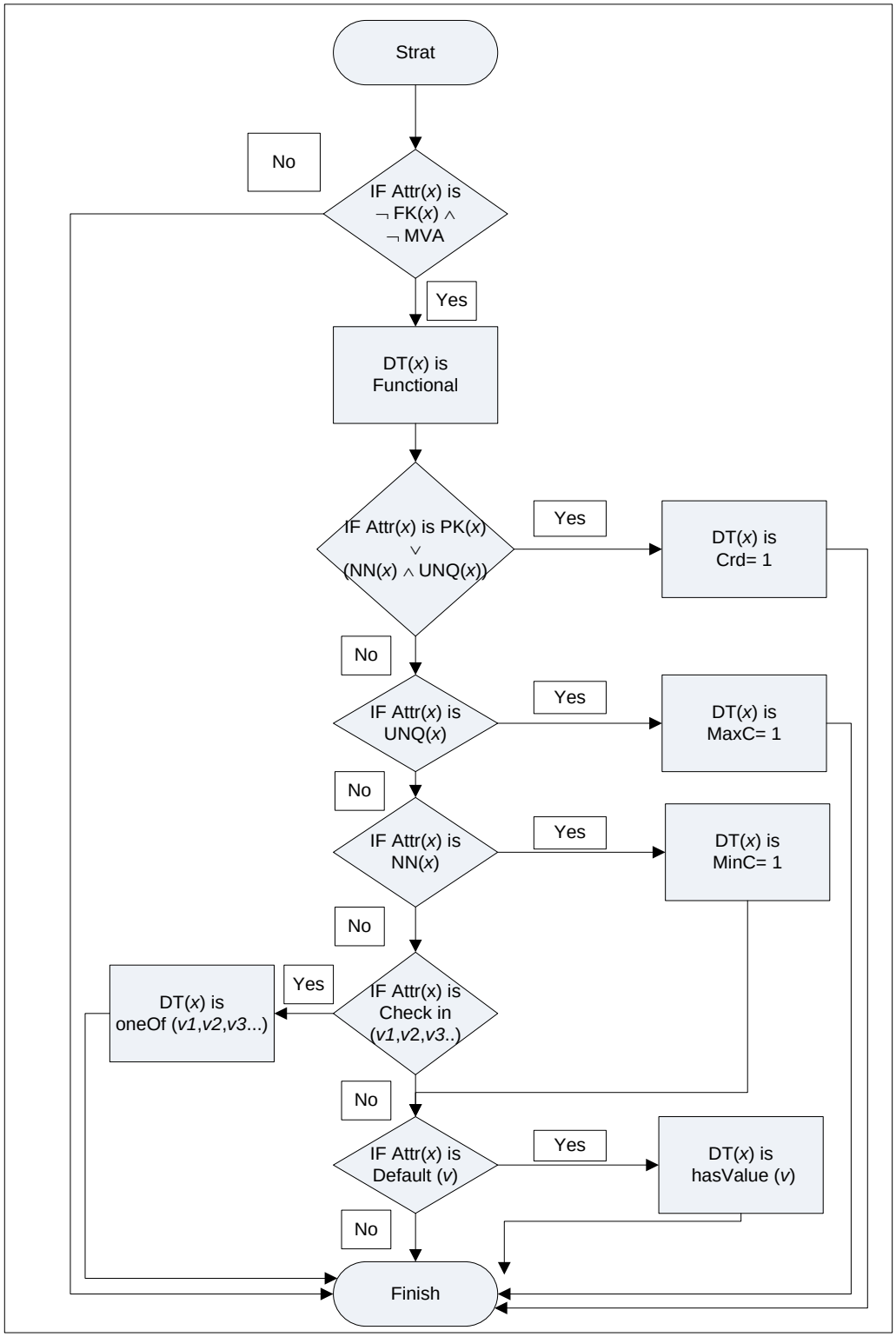


Figure 6-5: Data type Axioms Algorithm

- **Explanation:**

Figure 6.5 has shown the algorithm of applying SQL aspects. We created a data type for each attribute in each relation, except attribute that represented a FOREIGN KEY. In multivalued attribute a corresponding data type would be created, but without any axiom. We could subsequently annotate the data type with some axioms based on that obtained from the attribute constraints. Therefore if the attribute corresponding to the data type was representing a PRIMARY KEY we could assign the data type to cardinality of one. The same axiom was applied to an attribute with (NOT NULL and UNIQUE) constraints.

However if the attribute had the UNIQUE constraint we added to its corresponding data type *maximum cardinality* of one. For the attribute with NOT NULL constraint case the data type corresponding to that attribute would have *minimum cardinality* of one.

In case the attribute had a DEFAULT value, then the data type corresponding to it would have the functor *hasValue*. Whereas if there were many values to choose from for one attribute then the functor *oneof* would gloss to the data type. Any attribute could have more than one constraint such as NOT NULL and UNIQUE. However some constraints could not meet together for one attribute. For example the primary key or the attribute with UNIQUE constraint were neither a DEFAULT value nor CHECK IN constraint; whereas the attribute with NOT NULL constraint could have DEFAULT value but not CHECK IN constraint.

SQL-DDL is rich with dynamic features, such as triggers and assertions. Unfortunately, we are not able to convert these features to OWL ontology since the description of ontology is static.

6.4.5 Rules for the creation of object properties

There are two main types of OWL properties, object properties and data type properties. Object property is a relation that can connect an instance to another instance residing in two different classes in a particular direction, whereas data type properties connect an instance to a value with a specific XML Schema type. This subsection has presented the

rules of object properties creation as the rules of creating data type properties have been discussed aforementioned.

Indeed object properties in OWL are deemed similar to relationships in database. In fact, the general object property creation rule is that each relationship could be represented by a class with two pairs of inverse object properties or by only a pair of inverse object properties. In our approach, when we represented a relationship by an object property, which implicitly meant that we created an inverse of that property. However there were different relationships where our approach did not need to represent them by object properties such as:

- The relationship between vertical relations; since they integrated into one ontological class.
- The relationship between multivalued attribute relation and the owner relation; since both relations integrated into one class.
- The relationship representing the inheritance between two relations; since the *subclass* functor was adequate to represent this kind of relationship.

Before progressing, we have to distinguish between the degree of relationships and their cardinalities. For example, a relationship of degree one exists in one entity, and has been called a unary relationship; a relationship of degree two is a binary relationship between two entities etc. In terms of cardinality, relationships can therefore be of three types: One-To-One (1-1); One-To-Many (1-M) or Many-To-Many (N-M). We dealt with the One-To-One relationships with mandatory participation by either the fragmentation rule or the inheritance rule (IS-A relationship). Therefore Rule_ OP1 below has concerned the Many-To-Many relationship; whilst rules Rule_OP2 and Rule_OP3 apply to two different cases representing One-To-Many relationships, including N-ary relationships.

6.4.5.1 Rule OP1: Object property rules for binary relationships (Many-To-Many)

This rule was applied to relations built on top of a binary relationship relation with Many-To-Many cardinality. This rule declares the binary relationship relation state. If

there is a relation, R holds two foreign keys referring to other relations, for instance R_1 and R_2. Also, both R_1 and R_2 originally represent entities, not a relationship. It is noticeable that all attributes are part of the foreign keys, and each foreign key of R is a subset of the primary key of R. In an informal way, a Many-To-Many relationship is represented by a relation whose attributes relate between two entity relations, and the primary key of the Many-To-Many relationship is a concatenation of the two entity relations' primary key.

For the case of the binary relationship relation with additional attribute, the need of class creation was done through Rule_C4. The Rule_OP3 has considered the object properties creation for this case.

$$Rule\_op_1 \begin{cases} Object\ condition \\ \exists x, y, x', y' : \begin{pmatrix} (a)\ FK(x,R,x',R_1) \wedge FK(y,R,y',R_2) \wedge PK(x \cup y, R) \wedge \\ (b)\ x \cap y = \emptyset \wedge \forall z : (isFK(z,R) \Rightarrow z \in \{x,y\}) \wedge \\ (c)\ \forall t : (Attr(t,R) \Rightarrow t \in x \cup y) \end{pmatrix} \\ Object\ Action\ \ Create \\ \qquad\qquad 1) ObjP(OP_1, C_1, C_2) \\ \qquad\qquad 2) ObjP(OP_2, C_2, C_1) \\ \qquad\qquad 3) Inv(OP_1, OP_2) \\ \quad where \\ \qquad\qquad class(C_1) \ll (R_1) \\ \qquad\qquad class(C_2) \ll (R_2) \end{cases} \tag{14}$$

We notice here that there was no class creation to represent the relationship relation. Furthermore, the rule implies that Class (C_1) and Class (C_2) should already exist, thus ensuring the correctness of the relationship creation. Therefore, both the corresponding domain class and range class can represent the relationship with a pair of object properties.

6.4.5.2 Rule OP_2: Object property rules for a relation with unary relationship (self-relation)

There are two cases for the (One-To-Many) cardinality relationship. This rule was representative of the first case of object properties creation, and indicative of the unary relationship. The next section will discuss the second case.

For a relation R having a foreign key referencing the primary key of R itself, we created two inverse object properties, as shown in the condition and action below in Rule-OP_2. Creating two object properties was the natural way of representing the two sides of the relationship. Therefore all cases of the One-To-Many relationship were represented by two inverse object properties; regardless of the degree of the relationship.

For example (see Figure 6.6) the Manager of the Staff was also a staff member and therefore obvious that Manages was a unary relationship. For every staff member there was one manager; whilst at the same time the manager could be supervisor to many staff members. We thus created two inverse object properties for this relationship. The first object property was Superior, with *maximum cardinality* set to one or *Functional*, to ensure that each staff had only one manger. Whilst the second object property was Subordinate with no cardinality restriction.

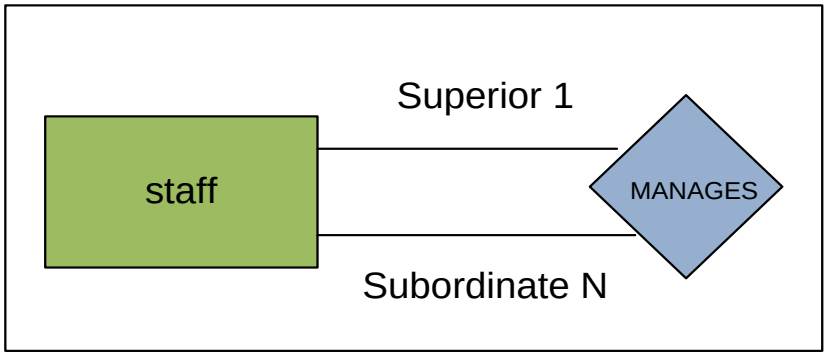


Figure 6-6: Unary relationships

$$Rule\_op_2 \begin{cases} \text{Object condition}: \\ \quad \exists x, y: \begin{pmatrix} (a)\ PK(x,R) \wedge \\ (b) FK(y,R,x,R) \wedge\ \mathrm{x} \bigcap \mathrm{y} = \emptyset \end{pmatrix} \\ \text{Object Action}:\ Create \\ \quad 1) ObjP(OP_1, C, C) \\ \quad 2) FP\ (OP_1) \\ \quad 3) ObjP(OP_2, C, C) \\ \quad 4)\ Inv(OP_1, OP_2) \\ where \\ \quad class\ (C) \ll (R) \end{cases} \tag{15}$$

6.4.5.3 Rule OP_3: Object default rule

For relations R_1 and R_2, if there was an attribute A part of R_1 referencing R_2, and regardless of whether A was part or not of the primary key of R_1, then an object property (OP_1) and its inverse object property (OP_2) could be created between the classes C_1 and C_2 corresponding to R_1 and R_2 respectively. The OP_1 domain was C_1 and the range C_2; and vice versa for OP_2. Indeed the default rule (Rule_OP_3) excluded both the self-relationship and Many-To-Many relationship.

$$Rule\_op_3 \begin{cases} \text{Object condition:} \\ \exists x, y : FK(x, R_1, y, R_2) \ \wedge \\ \neg\exists x, y, x', y' : \begin{pmatrix} (a)\ FK(x, R, x', R_1) \wedge FK(y, R, y', R_2) \wedge PK(x \cup y, R) \wedge \\ (b)\ x \cap y = \emptyset \wedge \forall z : (isFK(z, R) \Rightarrow z \in \{x, y\}) \wedge \\ (c)\ \forall t : (Attr(t, R) \Rightarrow t \in x \cup y) \end{pmatrix} \\ \text{Object Action: } Create \\ \qquad 1) ObjP(OP_1, C_1, C_2) \\ \qquad 2) ObjP(OP_2, C_2, C_1) \\ \qquad 3)\ Inv(OP_1, OP_2) \\ where \\ \qquad class(C_1) \ll (R_1) \\ \qquad class(C_2) \ll (R_2) \end{cases} \tag{16}$$

Some approaches such as [41] differentiated between a relationship existing between two relations R_1 and R_2, when an attribute A was a subset of the primary key of R_1 (weak entity case), referring to the primary key of R_2; or if A was disjoint from R_1 primary key. For the former case they created two object properties, OP_1 and OP_2; whereas the second case created just one object property, OP, with domain C_1 and range C_2. We chose to represent each relationship by a pair of inverse object properties because defining object properties in both directions is a practical choice, since each direction implies a specific role, and this allows more semantic detail to be held. One of the useful aspects of OWL is that it provided a term to represent the inverse of an object property; therefore, it is possible in OWL to create a pair of object properties and impose them to be inverses of each other.

This rule will also include the cases of:

- Binary relationships of cardinality (Many-To-Many) with additional attributes.
- Ternary and higher relationships.

In the case of the binary relationship (Many-To-Many) with additional attributes, we have already created a class to represent the binary relationship relation Rule_C4. The relationship will then be changed to two one-to-many relationships between the three classes, as shown in Figure. 6.7. Thus, in this case, the relationship has shown two pairs of inverse object properties with a class.

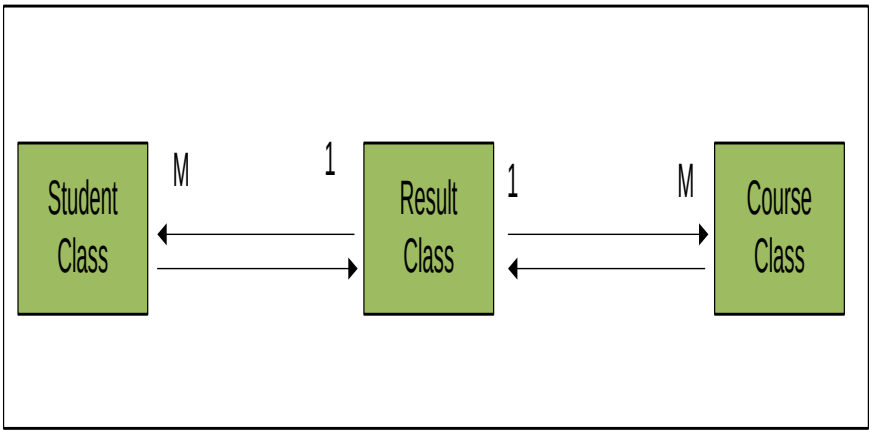


Figure 6-7: Binary Relationship with additional attribute

This rule will also treat a ternary or higher relationship in the same manner. For example, when we had a ternary relationship, we firstly created a class for it, using Rule_C4, then decomposed the ternary relationship to a binary one, before creating two inverse object properties between each of the corresponding classes. The number of object properties in the N-ary relationship will therefore be: N*(N-1).

Each object property in this case will also have such criteria:

- Inverse (OP_1, OP_2).
- $OP_1$and OP_2 *Cardinality* restriction set to one [50].

OWL allows the restriction of an object property by both *Functional* and *Inverse Functional* functors, thus ensuring the relationship of type one to one. Moreover,

imposing cardinality constraints to one allows the property to only have one instance. However OWL granted each instance with a unique identifier and therefore uniqueness between instances was not applicable; unless we used both *Functional* and *Inverse Functional* for the same object property.

- **Applying other SQL aspects to object property default rule**

There are many different cases which combine uniqueness and null restrictions into foreign keys. The following is the algorithm for applying them as depicted in Figure 6.8.

In database design, the foreign key refers to only one record of the range side relation; consequently its corresponding object property will be considered as Functional, and the inverse of the produced object property is inverse Functional However this generality was not applicable for object properties representing the (Many – To - Many) relationship; as each object property could have more than one instance in one time. Applying maximum cardinality restriction to inverse property is not applicable, since there might be an individual from the range class referenced by any member of the domain individuals.

6.4.5.4 General object properties characteristic algorithm :(17)

Before applying any of attributes characteristic for foreign keys we had to exclude all the foreign keys belonging to the binary relation of (Many-To-many) cardinality and a self- relation. The approach would then create two inverse object properties (OP_1, OP_2) representing each foreign key by Rule-OP_3.

i. If the attribute *x* is a FOREIGN KEY and NULL and not UNIQUE then make the OP_1 and OP_2 correspond to the attribute *x* with *Functional* (OP_1) and *Minimum Cardinality* $OP_2 = 0$.

ii. If the attribute *x* is a FOREIGN KEY and NOT NULL but not UNIQUE then make its *cardinality* (OP_1) *=1*, and *Minimum Cardinality* $OP_2 = 0$.

iii. If the attribute *x* is a FOREIGN KEY and NULL and UNIQUE then make it *Functional* (OP_1) and *Functional* (OP_2).

iv. If the attribute *x* is a FOREIGN KEY and NOT NULL and UNIQUE, but not a primary key then apply Rule_$C_{2nu.}$

v. If the attribute *x* is a FOREIGN KEY and part of the PRIMARY KEY for weak entity table or *N*-ary relations; or (Many-To-Many) relationship relation with additional attribute, then assign, *cardinality* (OP_1) *=1*,and *cardinality* (OP_2)*=1*

It is noticeable that the first four above rules prevent foreign keys from being part of the relation primary key. In fact, in order to fulfil that, we utilised the negation of UNIQUE or NOT NULL constraints.

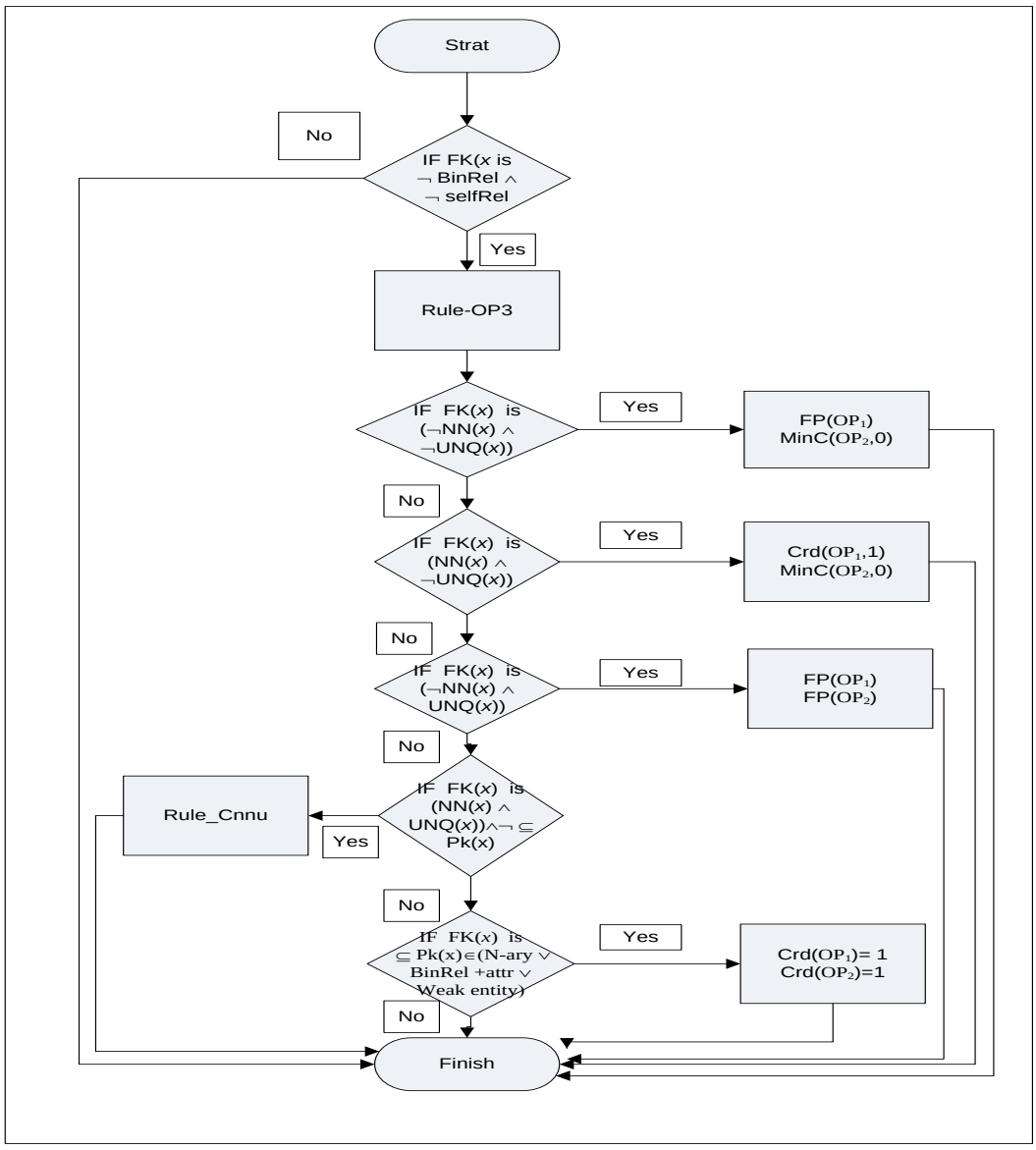


Figure 6-8: Object properties characteristic algorithm

- ***Discussion:***

Some property characteristics could be obtained from SQL terms such as *functional*, *inverse functional* and *inverse* of. However not all property characteristics could be obtained from SQL such as symmetric and transitive. Therefore domain experts should sometimes be involved in the designing decisions; especially when there are no obvious rules that could decide whether this characteristic was applicable or not for a property. For example an object property could have a symmetric characteristic or a transitive characteristic, both or none of them based on the specification of the relationship it represents.

Using a general rule could cause a design problem. For instance, Astrova *et al*'s approach [46] mapping all the One-To-One relationship to inheritance; or all unary relationships to object properties, with symmetrical characteristics. On occasion using specific examples would not show all the probabilities for representing property characteristics and may cause error in the modelling.

- **Symmetric Claim**

Astrova *et al*'s approach [46] claimed that all unary relationships can represent object properties with symmetric characteristic. However this generalisation is considered wrong.

- **Disproof**

To further explain this problem, three different examples are hence provided. The first example is university-course-offering with three tables (University table, Course table, and Offering table). Many universities often have the same course offered in different departments (the politics of this are to ensure departments receive student-credit-hour (SCH) numbers for their students taking a course in their department; even if it is the same as a course in another department). For example, if a university had a course "WS 310 "Black Women in America" offered in the Women's Studies Department, cross-listed as "AAS 310", and offered in the African-American Studies Department; and there was a History Course, "HIST 310" which was also cross-listed with the above

two; these would be taught in the same room, by the same instructor, with all of the same requirements.

The universities course offering table could be modelled as:

Offering Table instances:

Offering (O# = 1234, CNO = "WS 310"..., CONO = 5678)

Offering (O# = 5678, CNO = "AAS 310", ... , CONO = 9012)

Offering (O# = 9012, CNO = "HIST 310"... , CONO = 1234)

or as

Offering (O# = 1234, CNO = "WS 310"... , CONO = 5678)

Offering (O# = 5678, CNO = "AAS 310", ... , CONO = 1234)

Offering (O# = 9012, CNO = "HIST 310"... , CONO = 1234)

or even as

Offering (O# = 1234, CNO = "WS 310"... , CONO = 5678)

Offering (O# = 5678, CNO = "AAS 310", ... , CONO = 9012)

Offering (O# = 9012, CNO = "HIST 310"... , CONO = 5678)

...

All three of the above scenarios satisfy referential integrity and model the basic idea that these three course numbers form a set. The problem is that this cannot be differentiated syntactically from something like Person/Father table (the second example).

Person Table instances:

person(ID = 3456, Name = "Joe Smith", ... , FatherID =7890)

person(ID = 7890, Name = "John Smith", ... , FatherID = 2345)

person(ID = 2345, Name = "Paul Smith", ... , FatherID = NULL)

Or as a third example, consider a Parts table:

Parts Table instances:

Part(PartID = 6789, PName = "CPU", SubPartOfPart# = 0123)
Part(PartID = 0123, PName = "Motherboard", SubPartOfPart# = 4567)
Part(PartID = 4567, PName = "Computer", SubPartOfPart# = NULL)

In all these three examples, we have a self-relationship modelled in these tables. In all cases, the sample data given has maintained referential integrity, and therefore they are valid. However, for the purposes, they are thus modelling different concepts and the explanation has hence been given.

- The Offering example is symmetric ("if course A is offered with course B then course B is offered with course A"); and transitive ("If A is offered with B and B is offered with C then A is offered with C").
- The Parts example is not symmetric ("if A is a subpart of B, then B is a subpart of A" is FALSE); however, it is transitive ("If A is a subpart of B and B is a subpart of C, then A is a subpart of C").
- The Person/Father example is neither symmetric ("if A is the father of B, then B is the father of A" is FALSE); nor transitive ("If A is the father of B and B is the father of C, then A is the father of C" is FALSE).

Here the need to domain expert interferes is significant since it helps to decide whether characteristics are applicable or not.

In the area of deductive databases research this addressed problem could be solved. Tables deductive databases are used as in relational databases; however we augment them with rules written in Predicate Calculus that allow us to derive new information from the stored table. The rules could be used to define the types of restrictions under discussion.

For example, in the Person/Father example, we may write,

```
Father(X,Y) :- Person(X, ..., Y).
```

This means "X has Father Y if X appears as the first column of a tuple in the Person relation, and Y appears as the last". Certainly, this is true for any X and Y that meet the requirement. This would only be the rule for "Father".

For the Parts example, we may write,

```
HasSubPart(X,Y) :- Part(X, ..., Y).

HasSubPart(X,Y) :- HasSubPart(X,Z) , Part(Z, ..., Y).
```

Here, we had two ways of determining if X had Y as a subpart: one is the direct linkage, and the other is a recursive definition. The second rule allowed for transitivity.

For the offering example, we may add a third rule to model the symmetry,

```
OfferedWith(X,Y) :- OfferedWith (X, ..., Y).

OfferedWith(X,Y) :- OfferedWith(X,Z) , OfferedWith (Z, ..., Y).

OfferedWith(X,Y) :- OfferedWith(Y,X).
```

This highlighted the limitation of the way a self-relation is modelled in a relational model. Knowing a foreign key referring to the primary key in the same table would not help in deciding the property characteristic. The semantic of symmetric and transitivity need a domain expert to be obtained since they are same syntactically.

- **Transitive Claim**

The second claim for transitive case, Astrova et al.'s approach [46] considered the self-relation as transitive if the constraint of ON DELETE CASCADE applied.

- **Disproof:**

We used counterexamples that disproved the generality of using transitive with the self-relation characteristic.

The approach of [46] provided the example of the Spouse relationship representing self-relationship and it could be defined as symmetric object property. Conversely the Manager relationship was also a self-relationship from (Figure. 6.6) however it could not represent a symmetric object property.

For a transitive case, the Astrova *et al*. approach [46] only considered the self-relation as transitive if the constraint of ON DELETE CASCADE applied. And the transitivity rule said if X is related to Y and Y is related to Z then there was certainly an existent-dependent relationship between X and Y and subsequently a relationship between Y and Z. Applying “ON DELETE CASCADE” meant if X was deleted, Y would be deleted, and successively Z would be deleted, and that implied deleting X and consequently deleting to Z.

However, we could not be entirely sure if the relationship that was being modelled was always transitive. The typical example we used to illustrate a non-transitive relationship would be something like "Parent" in a genealogy database. Here, X could be the parent of Y, and Y the parent of Z; but X is not the "Parent" of Z, although “ON DELETE CASCADE” constraint was yet applicable for this case.

In general, the problem is whether self-relationships imply an ordering or subclass-type of relationship. The "Task/Sub-Task" example mentioned by the Astrova *et al.* approach [46] to support their claim was a subclass-type of relationship and transitive axiom is therefore suitable. However what if we had "Next-Task" instead of “Sub-Task”, specifying the next task to complete after this task was finished. In this case, X could have Y as its next task, and Y would have Z; but Z would not be the next task of X. The condition of “ON DELETE CASCADE” would be applicable here since the order of the task was important. Therefore deletion of the first task would imply deletion of the next task and eventually all the chain would be deleted.

From these two examples we substantiated that there was no guarantee of transitivity whilst using the clause ON DELETE CASCADE with self-relationship.

6.4.6 Rules for instances

This step was optional in our system because some web designers preferred that the data stayed in the database; which is more powerful for storing large-scale data sets than the ontology. However they needed a system to move queries from the global ontology website to the database tuples. Conversely others have preferred to create a knowledge base for the ontology produced by this rule in our system. We will discuss the advantages and disadvantages of both choices in the evaluation chapter.

We can use a two-step algorithm to migrate database tuples to ontological instances:

i. Each tuple is given a unique label name and migrated with all its data attributes except for the foreign keys.

ii. A natural join link between the label tables and the foreign keys to make instances for each object property with its corresponding individual.

For cases of fragmentation and multivalued attribute we needed to create a database VIEW to represent the integrated tables.

6.5 Summary

Our technical approach was presented in this chapter. We first specified the transformation assumptions. Then the predicates and the functions for the formal notation were explained. After that we discussed the transformation rules and demonstrated them by examples. The transformation rules depicted in a formal notation of the form IF condition THEN action. The class creation rules included the cases of fragmentation and inheritance. And we presented two solutions to solve the problem of fragmented tables. The first unique solution is by using different specification to represent the inheritance case and the fragmentation case. And the second solution is to analysis the number of records in each table in order to distinguish between the two cases. Also our approach is the only approach that can infer the multiple –inheritance

case from the logical model. Also we eliminate the problem of superfluous classes that produced from multi-valued attributes tables. Therefore from these new rules our approach gains its novelty. This chapter also explained the datatype creation rules and discussed the axioms obtained from SQL terms of attributes. Subsequently the rules of object properties were presented. These rules include cases of self relationship and Many-To-Many relationship and the binary relationship cases. Furthermore, we catch the cases where most the approaches are missed in their rules such as Many-To-Many relationship with additional attributes and binary relationship with additional attributes. In addition, our unique solution for higher degree relationship was discussed. Then this chapter disproved the claims of the symmetric or transitive characteristics of a self relationship. Finally this chapter presented the algorithm for migrating database instances.

CHAPTER SEVEN: TRANSLATING AN EXTENDED ENTITY RELATIONSHIP MODEL TO OWL ONTOLOGY

Objectives

- Building OWL ontology from an Extended Entity Relationship Model.
- Utilising an EER model in validating ontology produced from a RM model.
- Utilising an EER model in enhancing ontology produced from a RM model.

7.1 Introduction

This chapter informally provides general rules to generate OWL ontology from an EER model. It will explain how we can benefit from the existence of an EER for validating and enriching the OWL ontology produced automatically from the SQL source.

7.2 Translating extended entity relationship to OWL ontology

Since we consider both EER and ontological model are conceptual modelling [52], our focus in this section has only related to EER, as the ER model is a pure relational model. This means that the ER does not include advanced topics such as inheritance (specialisation/ generalisation).

Database design goes through two types of modelling. The first is the conceptual model (EER); and the second the RM model. Therefore, we can generate OWL ontology either from an Extended Entity Relationship model or a Relational model. This section has shown a way for producing OWL ontology from the conceptual model of the database EER. In fact we have suggested using EER as an ontology source immediately after

finishing the database conception stages, i.e. the new-born EER model is the best source from which to obtain the semantic since the actual database structure does not suffer from any modification.

7.2.1 Translating Method

The previous chapter concentrated on obtaining the ontological model from the relational model. However; in this chapter we have focused on two aims: Firstly, how to produce ontology from an extended entity relationship model, when the relational model was not available; and secondly how to utilise both database models to enhance the target ontology.

7.2.1.1 The general rule

The general rule for translating an EER model to an OWL model consisted of three steps:

i. Entities are represented by OWL classes.
ii. Relationships are translated into a pair of object properties; or two pairs of inverse object properties with a class, as may apply.
iii. Attributes can form datatype properties or a class with datatype properties, as may apply.

Indeed these steps were also applicable to an ER model since EER contained all the ER constructs. We therefore have to address an important point, which is the difference in defining names between the database and the ontology. In the database, any ER components are identified by their unique names; whereas for ontology, OWL components are identified by URIs.

The following subsections will present the method used to obtain ontology, definitions (classes, objects and datatypes) and their structure from the EER model in detail.

7.2.1.2 Creation of Classes

The process consisted of many steps:

- All entities (strong or weak) will be translated into an *owl: class* (see example 1(a, b) in Table 7.1).
- All composite attributes will be transformed into an *owl: class*(see example 2 (a, b) in Table 7.1).
- All *n*-ary relationships where *n* >2 will be transformed into an *owl: class*.
- All (Many-To-Many) relationships with supplementary attributes will form an *owl: class* (see example 5(a, b) in Table 7.1).

7.2.1.3 Datatypes creation

The process of producing datatypes obtained from entity attributes or relationship attributes is outlined below:

- All attributes of an entity will be converted to datatype properties. Moreover their domain will be the class corresponding to their owner entity. The range is their equivalent XML types. There are no constraints for the attributes in EER such as NOT NULL or UNIQUE, therefore we treat attributes as outlined:
    a. *Owl*: *Cardinality* equal to one for primary key, the alternative key and the discriminator attribute of a weak entity. In addition this restriction is applicable for all the attribute parts of a composite attribute.
    b. All other attributes will be assigned to the functor *owl*: *Functional*, except the multivalued attribute part of an entity or a complex attribute.
    c. Complex attribute is an arbitrary nested composite and multivalued attribute. We would not represent each internal composite attribute of the complex attribute here by a class; instead we represent its leaf node with *Functional* restriction, and for the multivalued attribute case we represent it without restriction. For example a person who could have more than one residence and each residence could have a single address and multiple phones. The composite attributes were shown between parentheses ( ); their components separated by commas and the multivalued between braces {}. The following example demonstrates a complex attribute:

{Address-phone ({Phone (Area-code, Phone-number)}, Address (House-number, Street-name, City, Postcode)}

- For relationship attributes, if the relationship is converted to a class, then each attribute belonging to this relationship will be assigned to the class corresponding to it as its domain and the equivalent XML types as its range. However there were two cases of a relationship that did not form a class:

  a. For (1-M) relationship, the attributes belonging to this type of relationship will be allocated to the class corresponding to the entity of the M-side as their domain.
  b. For (1-1) relationship, the attributes belonging to this type of relationship can be allocated to either class involved in the relationship.

7.2.1.4 Object properties creation

Each relationship is represented by a pair of object properties and this implicitly means they will be inverse properties of each other except the special cases. We have three cases of relationships:

- **Representing the Binary Relationship** (General Case):

  Each relationship will have pair of object properties each the inverse of the other; the domain and the range represented by the classes corresponds to the entities participating in this relationship.

  a. For (1-1) relationship both object properties will be Functional.
  b. For (1-M) relationship the 1-side is Functional.

- **Representing Self-Relationship**

  Self-relationship is a case of a binary relationship in which both the domain and range refer to the same entity. Object properties representing this relationship could use the class corresponding to the entity holding self- relationship as domain and range.

- **Binary Relationships between weak and strong entity**

  In this case the existence of an instance of a weak entity was based on the existence of its instance record in the strong entity. We thus created this relationship by an object with cardinality of one. The domain was the class corresponding to the weak entity; and the range was the class corresponding to the strong entity.

- **Binary Relationships (M-N) Cardinality (special case)**

  This relationship implied two relationships. We created only a single object property for each relationship without its inverse; and there would be only two object properties, instead of four object properties, for the general case. The first one would be assigned to the class corresponding to the M-side entity as its domain and the range is the class of N-side. The second object property would have the class corresponding to the N-side entity as its domain and the range as M-side class. Finally, the two objects would possess the inverse properties of each other.

- **Representing Relationships with class**

  Here we have many cases as illustrated:

  a. **Binary Relationships (M-N) with additional attributes**

  In this case there is a class and two relationships. One relationship is between the relationship class and the first entity class; and the second relationship between the relationship class and the second entity class.

  For the first relationship we created two inverse object properties. The first object used the class corresponding to the binary relationship class as its domain; the other class corresponding to first entity as its range and its inverse was vice versa with regards to the domain and range.

The second relationship was similar, treating with the other entity involved in this relationship. We created four object properties at the end and each two were inverse properties from each other (see Table 7.1(5.a, 5.b)).

b. **Higher Degree:**

Since each *n*-ary relationship was represented by class, we decomposed the *n*-ary relationships to binary relationships. Each binary relationship would then be represented by a pair of object properties known as inverse properties.

One object used the *n*-ary corresponding class as its domain and the range was the class corresponding to the entity participates in this binary relationship as its range; and the inverse object property was vice versa. This work would be completed for all n relationships. In addition we restricted all object properties considering the class corresponding to *n*-ary relationship as their domain to cardinality of one.

c. **Ternary relationship (special case)**

If the ternary relationship formed with (1- M- N) cardinality then we could decompose it into two relationships. The first relationship between M and N entities should be represented by a class. The (1-M) relationship is subsequently represented by a binary relationship between the new class and the class holding the 1 cardinality (see example 6.a and the translation in 6.b in Table 7.1 for more clarification). In the example, three entities were sharing one relationship (Supply). Two edges of the relationship were (Many), and the third (One). The best modelling for this case was to represent the part of the relationship holding (Many-To-Many) by a class.

In our example the combination of the primary keys of Part and Project entities formed the new class Project-Part. We subsequently treated the third relationship that held the one-side as treating the binary relationship of (1-M). Here the domain of the object property is the new class (Project-Part)

and the range is class corresponding to the entity of one-side (Supplier). Its inverse object is vice versa with respect to the domain and the range.

- **Representing a Relationship for Composite Attribute (special case):**

  This relationship does not exist in the original EER. However though the creation of a separate class for composite attribute we had to connect it with its owner entity. Therefore we created one object property between the class corresponding to the owner entity as the object property domain; and the class of composite attribute as its range (see 1.a and 1.b in Table 7.1).

- **Representing a Relationship for Complex Attribute (special case):**

  This relationship looked like the previous one. Here we created one object property between the class corresponding to the owner entity, as object property domain, and the class of complex attribute as its range(see 2.a and 2.b in Table 7.1).

- **Representing Inheritance**

  We had different cases of inheritance

  a. **Single Inheritance**

     For all class/subclass relationships (ISA) between two entities created superclass-subclass relationships (subsumption relationships) between their corresponding classes. This solution worked for specialist hierarchy cases whereby each entity had only one parent (class/subclass relationship).

  b. **Multiple Inheritance**

     We had to distinguish between hierarchy and multiple inheritance. The former represented specialisation hierarchy; whilst the later the specialisation lattice.

If the child entity had more than one parent entity, then the *owl: intersection of* term would make the shared sub-class inherit the characteristics of all its parent classes. This case thus represented the specialisation lattice.

c. **Specifying constraints on specializations**

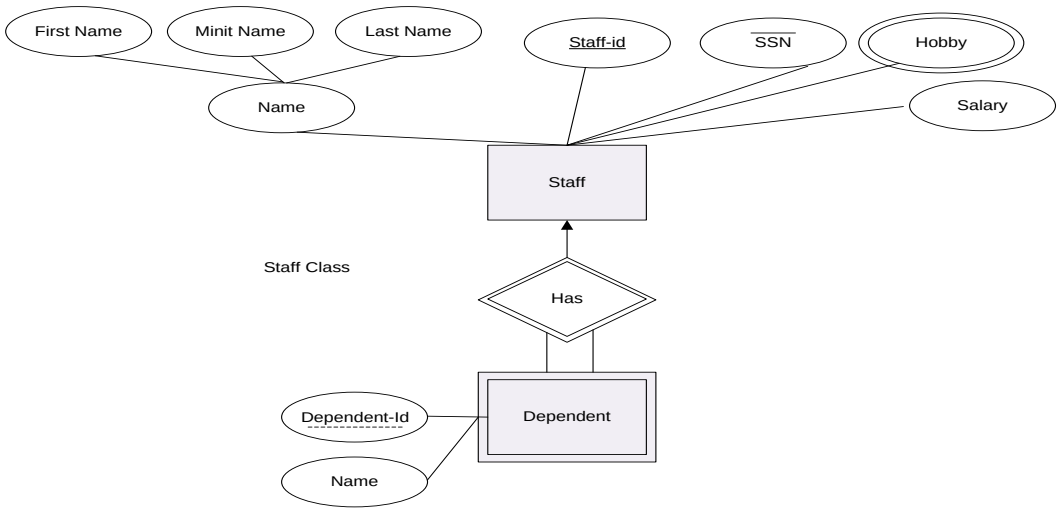
We can now represent the disjoint and complete constraints. The default of the *owl: subClassOf* construct is overlapping and partial. The disjoint and total constraints are thus outlined:

1. Disjoint is mapped to *owl: disjointWith* construct.
2. Total specialisation is mapped to *owl: unionOf* construct.

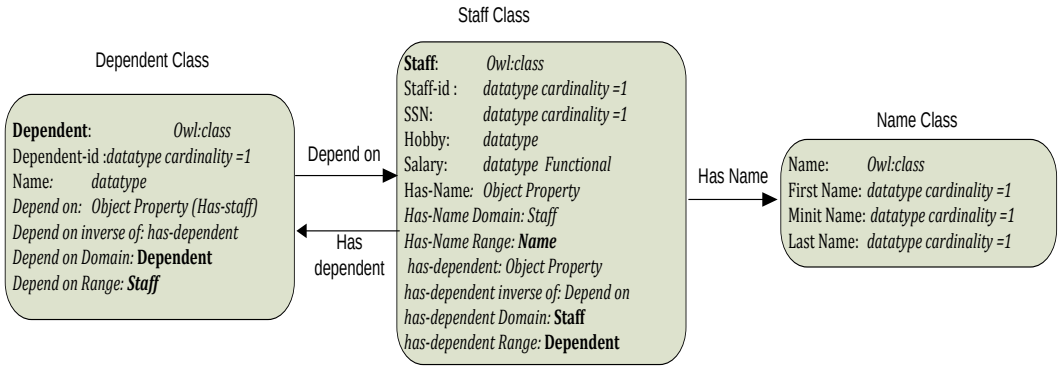
7.3 Examples

The examples in Table 7.1 contain a selection of cases from the EER model. It includes strong entity; weak entity; simple attribute; composite attribute; multivalued attribute and complex attribute. Furthermore there are examples for different kinds of relationship such as (1-M), (M-N), (1-M-N). And the cases corresponding modelling in OWL ontology are presented in the (b) rows.

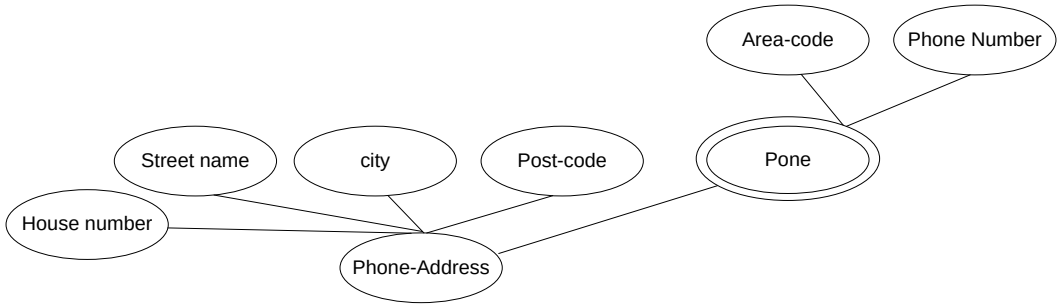
Table 7-1: Translating Examples from an EER to OWL Ontology

(1.a) Strong entity and weak entity with different attribute types


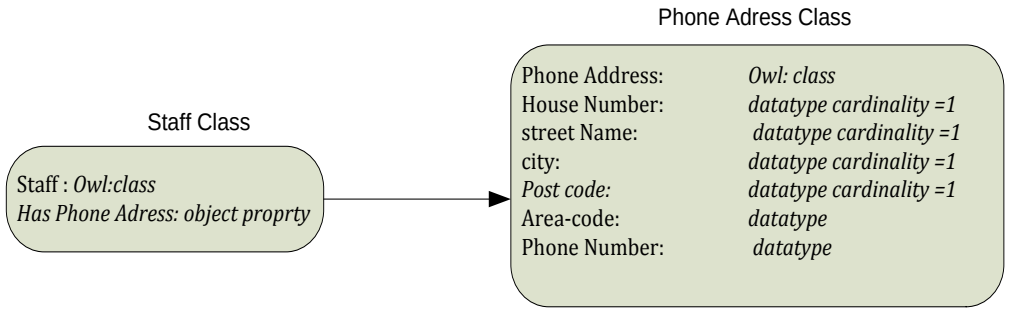
(1.b) Class and datatype representation in OWL for case (1.a)



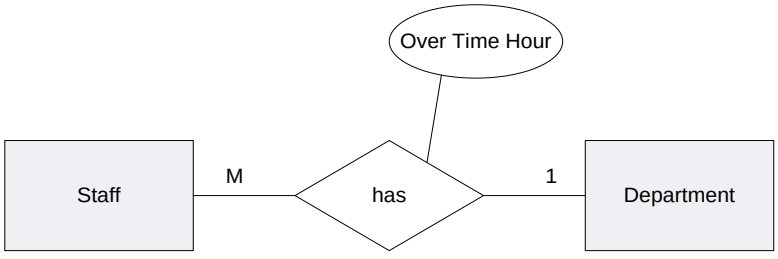
(2.a) Composite attribute



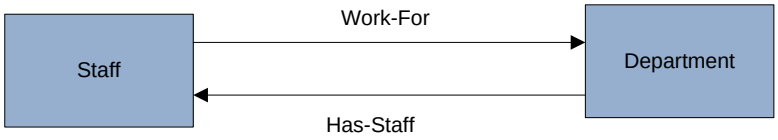
(2.b) Representing Composite attribute in OWL



(3.a) Binary Relationship (1-M) with attribute



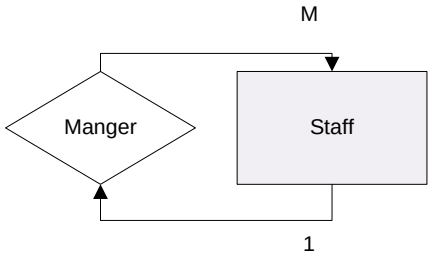
(3.b) Representing Relationship (1-M) with attribute in OWL



Staff:	*Owl:class*
Work-For:	*object property*
(Work-For) Inverse of :	*has-Staff*
Work-For Domain:	**Staff**
Work-For Range:	***Department***
Over Time Hour:	*Datatype*

Department:	*Owl:class*
Has-Staff:	*object property, Cardinality = 1*
(Has-Staff)Inverse of:	*Work-For*
Has-Staff Domain:	***Department***
Has-Staff Range:	**Staff**

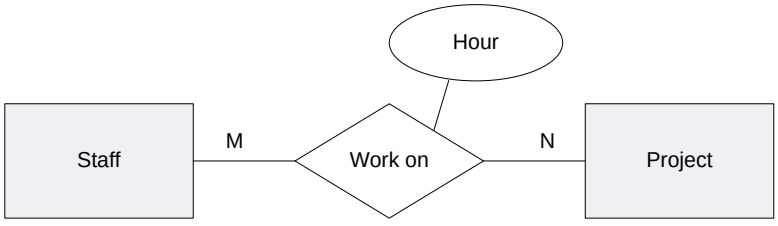
(4.a) Self relationship



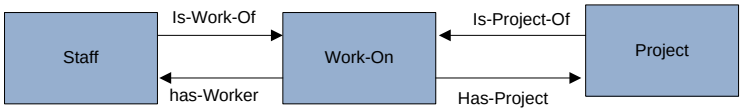
(4.b) Representing Self relationship in OWL

Staff:	*Owl:class*
HasManger:	*object property, Cardinality = 1*
(hasManger) Inverse of :	*Subordinate*
hasManger Domain:	**Staff**
hasManger Range:	**Staff**
hasSubordinate:	*object property*
hasSubordinate Domain:	**Staff**
hasSubordinate Range:	**Staff**

(5.a) Binary Relationship (M-N) with attribute



(5.b) Representing Binary Relationship (M-N) with attribute in OWL

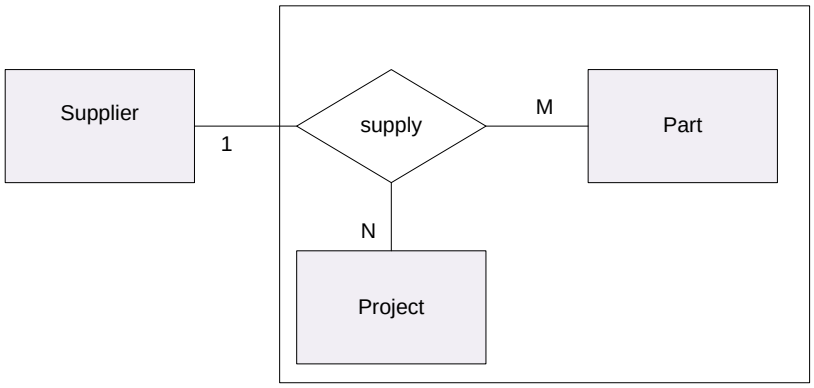


Staff:	*Owl: class*
Is-Work-For:	*object property*
(Is-Work-For) Inverse of :	*has-worker*
Is-Work-For Domain:	**Staff**
Is-Work-For Range:	***Work-On***

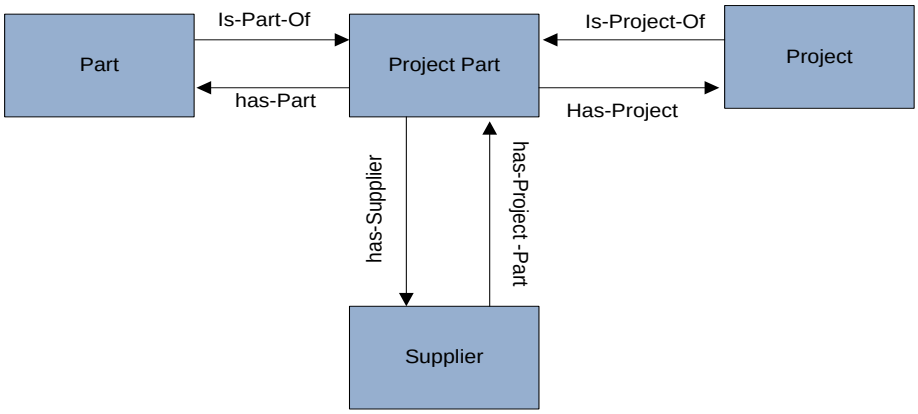
Work-On:	*Owl: class*
has-Worker:	*object property*
(has-Worker) Inverse of :	*has-worker*
has-Worker Domain:	***Work-On***
has-Worker Range:	**Staff**
Has-Project:	*object property*
(Has-Project) Inverse of :	*Is-Project-For*
Has-Project Domain:	***Work-On***
Has-Project Range:	**Project**
Over Time Hour:	*Datatype*

Project:	*Owl: class*
Is-Project-For:	*object property*
(Is-Project-For) Inverse of :	*has-Project*
Is-Project-For Domain:	**Project**
Is-Project-For Range:	***Work-On***

(6.a) Ternary relationship (1:M:N)



(6.b) Representing Ternary Relationship (1:M:N) in OWL



7.4 The Algorithms of Translating an EER to OWL Ontology

The algorithms of creating OWL ontology from an Extended Entity Relationship model contained two steps. Firstly, to deal with the entities and attributes (see Figure 7.1); and secondly, to handle the relationships. Before we carried out these steps there was one requirement in order to ensure successful implementation of fixing the model representation; since EER was a graphical representation and it was difficult to parse unless we reformatted it into a formal definition. Therefore we chose to present the EER diagram in terms of facts (schema way) which were easy to parse.

The second option was to obtain the schema from the EER diagram through tool packages. Using EER specification packages (CASE tool) such as Power Designer was not preferable, since these tools did not support all the semantics of the EER model. For example Power Designer did not currently support attributes on relationships or higher degree relationships [47]. Therefore, we chose the first choice in order to apply the two algorithms of creating ontologies.

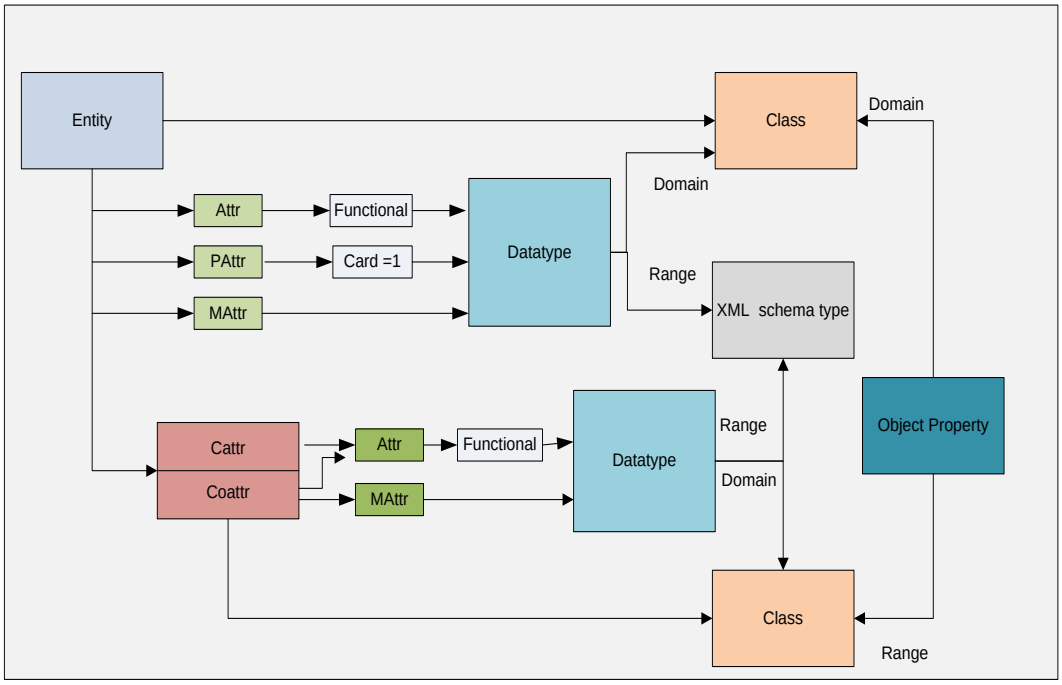


Figure 7-1: Translating Entity and Attributes of the EER model to OWL Ontology

7.4.1 Notations

Before the translating can take place, the EER diagram should be converted into a set of formal definitions or facts. For example the EER schema is a means of facts representation, however representing the model only by a schema would not help in designing the ontology. Therefore we appended some notations to the EER schema constructs in order to add semantic to the EER model.

We used the idea of prefixing the names of database constructs in the follow manner:

- *E:* for strong and weak Entity;
- *R* Relationship;
- *Attr:* a set of single attributes of entity or relationship;
- *Pattr:* is primary key attribute;
- *Alattr:* set of alternative key attributes;
- *Cattr:* set composite attributes;
- *Coattr:* set complex attributes;
- *Mattr* : set of multivalued attributes;
- *Crd :* the E:card is a cardinality ratio of relationship toward a specific entity;
- *ISA:* represent inheritance relationship and
- *No:* number of.

We suffixed each attribute with its type:

- *Type:* the *A: Type* the domain of all possible values of an Attribute, i.e. integer, string etc.

7.4.2 EER model structure rules

- Each entity could have any number of attributes, including multivalued composite and complex attributes.
- For the keys of an entity, only one primary key is allowed. This primary key can be represented be either simple or composite attributes.

- The entity can have any number of alternative keys.

This means:

- For an entity $E \in E_s$ such that $Attr \in (E)$ = [..., A : *Type*, ...],Where A is the attribute name, and *Type* is the type domain of the attribute such as integer. In addition the A includes simple and *(Mattr)* multivalued attributes.
- For a composite attribute of an entity $Catter \in E$ such that *(Catter)* = [..., A : *Type*, ...].
- For complex attributes of an entity $Coattr \in E$ such that *(Coattr)=* [..., A : *Type*, ...] + *Atts (Catter).*
- For an entity $E \in E_s$ such that $\exists\ Pattr \in E_s \wedge |Pattr| = 1$. Also $Pattr \in Catter \vee (A \in Attr)$.

Noticeably there was no need to represent the derived attribute since it was a simple type attribute.

7.4.3 Class and Datatype Creation Algorithm

Table 7.2 shows the Algorithm 7.1 and its sub-functions. This algorithm has demonstrated all the possible cases for an entity with its different kinds of attributes. In addition it was written in Pseudo Code and could be implemented with any programming language. In accordance with the type of the attribute, the input for each sub-function varied. The algorithm presented took one entity at a time, with its own attributes; and subsequently for each attribute that had the prefix of $attr$ create the datatype corresponding to it with *Functional* constraint (see Function 1). However, if the prefix was *Mattr* then only create the datatype with its class domain corresponding to its own entity without a restriction (see Function 2).

For the composite attribute each component would be a datatype with cardinality of one. For the primary key, there were two cases. The first case was a primary key as a simple attribute; however it was restricted with cardinality of one, instead of *Functional*, and the alternative key was handled as this case of primary key as well. The second case was the primary key as a composite attribute; and we called the composite attribute

function to handle it. As for a complex attribute, which may have contained multivalued or composite attributes, we created only one class to represent it; and for the internal composite attribute we also created a class. The other levels of nested composite attributes the algorithm considered only the leaf nodes.

Table 7-2: Algorithm 7.1(entity and attributes representation)

Algorithm 7.1(entity and attributes representation)
Input: entity and its attributes of the EER **Output: class and datatypes creation** **Steps:** ***For all*** *E.name create class.name with* ***For all*** *A ∈ E then* ***if*** *A ∈ attr then fuction1 (E, A);* ***Else if*** *A ∈ (Pattr or Alattr) then fuction2 (E, A);* ***Else if*** *A∈ Mattr then fuction3 (E, A);* ***Else if*** *A = Cattr then fuction4 (E, A);* ***Else*** *fuction5(E, A);* ***Endif;*** ***Endif;*** ***Endif;*** ***Endif; EndFor;*** ***EndFor;*** **End Algorithm.**
Function 1 `/* for normal attribute */`

Input: entity name E and attribute name A
Output: datatype creation
Steps:
For all attr.name: Type then
datatype(**A**) functional (**A**), domain(E) , range (Type (**A**))
EndfFor;
End;

Function 2 `/* for primary or alternative key attribute */`

Input: entity name E and (primary key or alternative key A)
Output: datatype creation
Steps:
If (Pattr.name: Type or *Alattr.* name: Type ∈Attr then let
Datatype (**A**) cardinality =1, domain (E), range (Type (**A**));
Else call function4;
EndfiF;
End;

Function 3 `/* for multivalued attribute*/`
Input: entity name E and multivalued attribute A **Output: datatype creation** **Steps:** For all Mattr.name: Type then

datatype(**A**) domain(E) , range (Type (**A**));
EndFor;
End;

Function 4
/* for composite attribute */

Input: entity name E and composite attribute X, attributes A
Output: datatype creation, class creation
Steps:
Create class for X
For all attr(A) ∈ Cattr(x) then
Datatype (A) cardinality =1, domain (x), range (Type (A));
EndFor;
Create an object properties OP (has-x)
OP(has-x) functional, OP(has-x) domain E, OP (has-x) range X;
End;

Function 5
/* for complex attribute */

Input: entity name E and complex attribute X, composite attribute Z, attributes A, multivalued attribute M

Output: datatype creation, class creation
Steps:
Create class for *X*
For all attr(A) ∈ *Coattr(x)* then
Datatype (A) cardinality =1, domain (X), range (Type (A));
EndFor;
For all Mattr(M) ∈ *Coattr* then
Datatype (m), domain (X), range (Type (m));
EndFor;
For all Cattr (Z) ∈ *Coattr* then
Call function 4(X,Z);
EndFor;
Create an object properties OP (has-x)
OP(has-x) functional, OP(has-x) domain E, OP (has-x) range X;
End;

7.4.4 Relationships

Each relationship could connect between any numbers of entities. This number represented the degree of the relationship. Each entity participate in any relationship would have one cardinality ratio of type one or many. Moreover the relationship could have any number of attributes. Finally the special case of the (One-To-One) relationship was the inheritance relationship (ISA).Thus meaning:

- For a relationship $R \in R_s$ ((... ,E_i :crd ,..,)∧(..., $Attr_j(A)$: type(A), ...) ∨, E1 is-a E2) where i = 2..n, j=0..n.

The degree of the relationship would have one type of (unary, binary, or *n*-ary) relationship. In order to specify the degree of a relationship we counted the distinct number of entities participates in this relationship. Moreover the special type of a binary relationship was the ISA relationship, which linked the two entities with the (One-To-One) cardinality ratio.

Table 7.4 has shown Algorithm 7.2 and its sub-functions for relationships representations in the ontological model. It has considered both the cardinality of the entities and the degree of the relationship. Noticeably E represents both the entity in the EER model and its corresponding class in the ontology model. We had eleven cases to represent a relationship in the EER model which Table 7.3 has demonstrated.

Table 7-3: Algorithm 7.2(Relationship Representation)

Relationship	cardinality ratio	Relationship attribute	Function
Unary	(1-1)		Function 5
Unary	(1-M)		Function 6
Binary	(1-1)	No	Function 7
Binary	(1-1)	Yes	Function 7 and Function 1
Binary	(1-M)	No	Function 8
Binary	(1-M)	Yes	Function 8 and Function 1
Binary	(M-N)	No	Function 9
Binary	(M-N)	Yes	Function 10
N-ary		No	Function 11
N-ary		Yes	Function 11 and Function 1
ISA			Function 12

We used Function one from Algorithm 7.1 to create the functional attributes. In addition each *n*-ary relationship was decomposed to binary relationships.

Table 7-4: Algorithm 7.2(Relationship Representation)

Algorithm 7.2(Relationship Representation)

Input: relationship with its entities names and their cardinalities ratio
Output: object property creation
Steps:

```
For all R Do
    If degree of R =1 ∧ crd1 =1 ∧ crd2 = 1
        Then Function 5;
    EndIf;
    Else If degree of R =1 ∧ crd1 =1 ∧ crd2 = M
        Then Function 6;
    EndIf;
```

Else if degree of R =2 ∧ crd1 =1 ∧ crd2 = 1 ∧No(attr)= 0
Then Function 7;
Else if degree of R =2 ∧ crd1 =1 ∧ crd2 = 1 ∧No(attr) ≥ 1
Then Function 7;
For all
A∈ R Function 1(E1,attr, attr:type);
EndFor;
Else if degree of R =2 ∧ crd1 =1 ∧ crd2 = M ∧No(attr) = 0
Then Function 8;
Else if degree of R =2 ∧ crd1 =1 ∧ crd2 = M ∧No(attr) ≥ 1
Then Function 8;
For all
A ∈ R Function 1(E2,attr, attr:type);
EndFor;
EndIf;
Else if degree of R =2 ∧ crd1 =M ∧ crd2 = N ∧ No(attr)=0
Then Function 9;
EndIf;
Else if degree of R =2 ∧ crd1 =M ∧ crd2 = N ∧ No(attr) ≥1
Then Function 10
For all
A ∈ R Function 1(R,attr, attr:type);
EndFor;
EndIf;
Else if degree of R >2 ∧ No(attr) =0
Then Function 11;
Else if degree of R >2 ∧ No(attr) ≥1
Function 11;
For all
A ∈ R Function 1(R,attr, attr:type);
EndFor;
Endif;
Else if R= isa
Then Function 12;
EndIf;
Endfor;
End Algorithm;

Function 5
`/* self relationship cardinality (1-1)*/`

Input: relationship R, cardinality crd1=1,crd 2=1, entities names E
Output: object property creation
Steps:
Create 2 inverse OP1, OP2:
OP1 domain E Range E card= 1;
OP2 domain E Range E card= 1;
End;

Function 6
`/* self relationship cardinality (1-M)*/`

Input: relationship R, cardinality crd1 =1,crd 2= M, entities names E
Output: object property creation
Steps:

Create 2 inverse OP1, OP2:
OP1 domain E Range E card= 1,
OP2 domain E Range E;
End;

Function 7
/* binary relationship cardinality (1-1) */

Input: relationship R, cardinality crd1=1,crd2=1, entities names E1,E2
Output: object property creation
Steps:
Create 2 inverse OP1, OP2:
OP1 domain E1 Range E2 card= 1;
OP2 domain E2 Range E1 card= 1;
End;

Function 8
/* binary relationship cardinality (1-M) */

Input: relationship R, cardinality crd1=1,crd 2=M, entities names E1,E2
Output: object property creation
Steps:
Create 2 inverse OP1, OP2: OP1 domain E1 Range E2 card= 1,
OP2 domain E2 Range E1;
End;

Function 9
/* binary relationship cardinality (M-N)*/

Input: relationship R, cardinality crd1=M, crd2=N, entities names E1,E2
Output: object property creation
Steps:
Create 2 inverse OP1, OP2: OP1 domain E1 Range E2,
OP2 domain E2 Range E1;
End;

Function 10
/* binary relationship cardinality (M-N) with attribute(s) */

Input: relationship R, cardinality crd1=M, crd2=N, entities names E1,E2
Output: object property creation, class creation
Steps:
Create class R
Create 2 inverse OP1, OP2: OP1 domain E1 Range R crd =1,
OP2 domain R Range E1 crd =1;
Create 2 inverse OP3, OP4: OP1 domain E2 Range R crd =1,
OP2 domain R Range E2 crd =1;
End;

Function 11
/* higher degree relationship */

Input: relationship R, cardinality crd1, crd 2, entities names E1,E2, …En
Output: object property creation, class creation
Steps:
Create class R
For i=1 to degree
Create 2 inverse OPi, invOPi: OPi domain Ei Range R crd =1,
invOPi domain R Range Ei card =1;
EndFor;
End;

Function 12

```
/* binary relationship of is-a type*/
```
Input: relationship, cardinality, entities names E1,E2
Output: object property creation, class creation
Steps:
Create E1subclass E2
/* Note: E1 and E2 classes do exist */
End;

7.5 Advantages and Disadvantages of EER as an Ontology source

There are many advantages in utilising an EER model as the main source of our approach compared with the RM model. These have thus been demonstrated:

i. Clear hierarchical tree which show the disjoint between entities and the shared sub-class of multiple inheritance.
ii. Easy to identify the kind of attributes such as composite or multivalued etc.
iii. Easy to generalise group of entity sets to one super entity.
iv. Naturally translating EER models to ontology preserves more information [36], by the distinction between entities and relationships.
v. Explicit participation and cardinality of relationships.

Both [6] and [36] claimed that the EER model was better for producing ontology than the RM for the following reasons:

- Since mapping from an EER model to the RM includes losing some semantics. As a result the RM is poorer in semantics than the EER.
- Since both the EER model and ontological model share many characteristics of conceptual modelling.

Due to these reasons translating from the EER model to ontological model would preserve more semantics. At this point this argument is partially true. However there are many disadvantages of using the EER model which would disprove the above claim such as:

i. The model is difficult to parse since it is a graphical model. Obtaining all the information from the EER diagram and entering such into a program to produce a facts model is a monotonous and error-prone process.
ii. There are three possibilities which reflect the failure of the EER model in representing the current database:

    a. There was no conceptual model created during the process of generating the database.
    b. Creating the conceptual model was only achieved in the first stages; i.e. the evolution of the database which included many modifications supposed to be documented and not fulfilled. Therefore the EER model was not updated. In reality many of these modifications change at the implementation level usually do not take place in the original EER.
    c. Designers create conceptual models without much effort in the designing process, since they keep their focus on producing the logical model of the database. Therefore the EER model might not present the real implementation of the database.

iii. The model is available during the conception of the database however it is now lost.
iv. The available database conceptual modelling tools do not support all the characteristics of an EER.

Moreover, the EER model was unable to create a full ontology because of the following factors:

i. There are no explicit declarations of datatypes (domains of the attributes). Therefore the database designers have to add them manually. There are no attribute constraints such as NOT NULL and UNIQUE.
ii. Some attribute characteristics are unavailable such as (symmetric, transitive and enumerated).
iii. There is no value restriction (i.e., no restrictions like, “age is a positive integer”).

iv. There are no instances. Consequently we would not be able to produce a knowledge base.

If we consider the disadvantages we specifically mentioned, the absence of the EER occurs in many databases. We found the above argument for an EER model producing better ontology than the RM which is weak.

If both an EER model and RM were available which source would we start with? Why?

There are two possibilities, either starting with an EER model then subsequently an RM, or vice versa. Indeed we have to start with the richer semantic. However there are no obvious criteria to specify which model would be better in designing the ontology. Logically we have to start with the original model (EER). However our approach does not start from the EER model as expected, as other approaches have claimed it to be easier for obtaining information.

However we started with the RM; since the process could be achieved in a fully automatic way as SQL-DDL is the source, whereas it was not possible to automate the complete process of the EER model due to the graphical representation which makes it difficult to parse. Moreover the EER model per se could not produce a complete ontology; whilst conversely the RM model was able to.

Other reasons that effected our decision were the availability of the SQL-DDL in each database making it easy to obtain; and finally it reflected the current database. By considering both the advantages and the disadvantages of EER and RM we decided to gain the benefit from both sources, if indeed both existed. We therefore suggested producing the initial ontology automatically from the RM, then using the EER model as a validation and enhancing method in order to catch additional or missing semantics.

7.6 Validation and enhancing

The previous chapter concentrated upon producing OWL ontology from the logical model written in SQL-DDL statements; whereas the first part of this chapter has focused on generating OWL ontology from the conceptual model of database i.e. EER

model. In order to utilise the two sources of the database, we suggest using the RM to produce the initial ontology; and subsequently using some aspects of the EER in both the validating and enhancing procedures.

In Section (7.6.1) we will explain the procedure of validating steps; whereas the next section will describe the procedure for enhancing the steps.

Before both validating and enhancing processes can take place we have to ensure that the EER model is representing the current real (updated) database. This is the responsibility of the database designer to check if both the RM and the EER model share the same concepts. In case, there is a lack of constancy between the two models, the database designer could analyse the RM to obtain the undocumented concepts.

The analysis here does not seek to produce a complete conceptual model for the database by using reverse engineering, but rather to extract some information which could be utilised in the validation process. Indeed requiring the involvement of the database designer is considered very important since they would be responsible for applying both validation and enhancement procedures. Two procedures demand a database designer and not an ontology expert for the following reasons:

- Since the analysis will be conducted on a database model.
- Applying the two procedures can be handled by a database designer even if they are not aware of ontology design.
- The numbers of available database designers are greater when compared with those in ontology.

7.6.1 Validation Procedure

We have to validate the ontology model to ensure the correctness of the transformation method. This means all the conceptual parts of the database should correctly correspond in the ontology.

The validation contained two stages. The first stage was to locate the concerns which may have led to incorrect ontology representations. Such as:

i. Multivalued attributes.
ii. Shared entities.
iii. Class hierarchy.
iv. Relationship of (1-1).

These concerns could easily be obtained from an EER model. All entities in the EER model were now represented by tables in the RM model with the exception of the vertical partitioning for tables. To verify the correctness of this mapping between the EER model and the Relational model we accumulated the number of all entities; higher degree relationships; (Many-To-Many) relationships and multivalued attributes, and this number should have matched the number of tables. Otherwise there were fragmentation tables for some entities. The database designer had to specify the superfluous tables and their crossholdings at this stage.

The second stage involved the database designer starting to refine the produced ontology by following the outlines validation steps:

i. All fragmentation tables of an entity should be represented by a class.
ii. Check that the multivalued attribute is represented by datatype property with no restriction i.e. no class, and no functional restriction. Multivalued attribute merges with its owner class.
iii. For all IS-A relationship there will be a subclass relationship.
iv. Check the subclass hierarchy; which includes tracking the IS-A chain between classes.
v. For the One-To-One relationship, there is no subclass relationship, instead treated as object properties.
vi. The RM model specify the shared entity indirectly, therefore the database designer makes sure to catch the multiple parent inheritance.

7.6.2 Enriching the Ontology

There were two ways to enhance the ontology produced. One of them was by extracting some semantic from the EER model, whilst the other was based on the experience of the database designer.

7.6.2.1 Analysing the EER Model

This procedure included obtaining the additional context of the relational model in order to enrich the target ontology. The remaining semantic features that were the responsibility of the domain expert; to be obtained manually from the EER model are hence outlined:

- If an entity contains a composite or a complex attribute, the best modelling for this case is to build a new class corresponding to it; and subsequently adding an object property to link between the new class of composite or a complex attribute and its owner class. This was similar to the case carried out with the Name attribute in Table 7.1 (1.b).
- The database designer adds cardinality participation for each object property, because the cardinality restrictions cannot be obtained from SQL.
- N-ary relationship with cardinality ratio of (1-M-N) can be partially represented as a class for the relationship of (M- N) cardinality and the other part of relationship (1- M) side can be represented by object properties.

7.6.2.2 General enhancing advice

i. Provide proper names to classes, object and datatype properties since most database designers use abbreviations in naming database components.
ii. Specify the quantifier restrictions such as "*allValuesFrom*" or "*someValuesFrom*" to properties, depending on the semantics of these properties, since these restrictions could not infer from the SQL statements or the EER diagram.
iii. Augment the relationship with the characteristics achieved such as *transitive* or *symmetric* or both of them as the case may be.
iv. Generalise the classes that share common attributes by superclass.

7.7 Summary

This chapter focuses on two parts. The first part is the translation system from EER model to OWL ontological model. In this part, the general method was explained, and

the rules of class and datatype and object properties are discussed. Here we emphasis on the concepts that are not available in the logical model such as composite attribute, complex attribute and multiple inheritance. Then the examples take place in order to explain the graphical representation of the EER model and its equivalent in OWL syntax. After that, this chapter provided the algorithms written in Pseudo Code for creating all the ontology elements. The final section of the first part showed the reasons behind choosing the logical model over the conceptual model. The second part of this chapter focused in using the EER model to enhance the ontology produced by the logical model. Also it provided the validation steps for the ontology produced by the logical model through utilising the semantics obtained from the EER model.

CHAPTER EIGHT: CASE STUDY AND PROTOTYPE IMPLEMENTATION

Objectives

- Apply all the rules from Chapter 6 to a complete example.
- Explain the prototype features.

8.1 Introduction

This chapter shows all the stages for generating OWL ontology. It begins with an analysis of the database. This step is followed by the production of the Extend Entity Relationship model and subsequently converting the EER model to the Relational model scripted in SQL-DDL. Ultimately the SQL will be transformed to OWL ontology.

8.2 University database example

For our model example we considered capturing a variety of modelling choices. Moreover our expository University example provided many different cases that typically existed in the real database. For instance our example included strong and weak entities; vertical partitioning of tables; multivalued attributes; composite attributes; the IS-A relationship; multiple inheritance; and all different relationships with a cardinality ratio (One-To-One; One-To-Many and Many-To-Many). Moreover the University example included different degrees of relationship such as unary (self relationship), binary and ternary relationships.

The University database has stored information about students and courses offered by a specific department for particular degree programs. However for simplicity, the example would not cover the entire University system. Therefore we excluded the collage entity and considered the module entity as attributes because there was no need

to enlarge the database with repetition of cases. From another standpoint the database was also considered for the purpose of storing the faculty member's information, including academic staff. Our database example illustrates part of the University database, which mainly concentrates on the student and staff entities and their relationships.

8.2.1 University database requirements

The following are the database requirements suggested from the analysis phase:

8.2.1.1 University entity

- The university is organised into colleges, each with many departments. Each department has a unique department number, department name, phone number and a particular staff member who works for the department.
- The department offers many different courses, each of which has a unique course number and course name and course credit hour.
- We keep track of the staff information including full staff name, date of birth, address, email and sex.
- The academic staff are all members of the college workforce. Besides storing staff information, we also store information regarding their present post, one of which must be of this type:
  ('Teaching Assistance', 'Instructor', 'Assistant Professor', 'Associate Professor', 'Professor') and their specialty, and national identify number.
- The database is also keeping track of staff dependent names and their staff relationships.
- For the student stores information the name, student identification number, address, sex and national identify number are required. Moreover each student has many hobbies.
- The graduate students' include Master and PhD students. In addition to student attributes they have a research field attribute.

8.2.1.2 University relationships

- Staff and students, with specific major modules, work and study within each department. Furthermore during each semester the department offers many courses.
- Each member of staff must work for one department and has a specific manager. In contrast the manager can supervise more than one staff member.
- The staff might have many dependents such as a spouse or children.
- An academic staff member is also a member of staff. Academic members of staff teach many different courses.
- Two offered courses could be co-offered, and recorded as a self-relation in the course entity set.
- A student registers in each semester for many courses; and the course therefore has many students. At the end of the semester the students acquire grades for their course.
- Each section has a student register associated with it; academic staff who teach, and a course which will be offered.
- Graduate students who study Master or PhD qualifications need not have a job in the university.
- The post-graduate student can help in the teaching of some courses. The post is considered that of 'Teaching Assistance'.

We postponed adding some attributes to the initial database. We subsequently used them to show their effects on the model.

8.2.2 University EER diagram

The EER diagram represents all the component of the conceptual model of the database. Here is the summary of the ER diagram notation:

- **Rectangles** represent entity sets.
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets; and entity sets to relationship sets.

- **Line with triangle head** represent the (IS-A) relationship.
- **Ellipses** represent attributes.
    - **Double ellipses** represent multivalued attributes.
    - **Dashed ellipses** denote derived attributes.
    - **Underline** indicates primary key attributes.

Figure 8.1 shows the corresponding entity relationship diagram for our university example. Here we only focused on the entity and the relationships. The diagram also shows the cardinality ratio. However for simplicity the diagram has eliminated the attributes of entities.

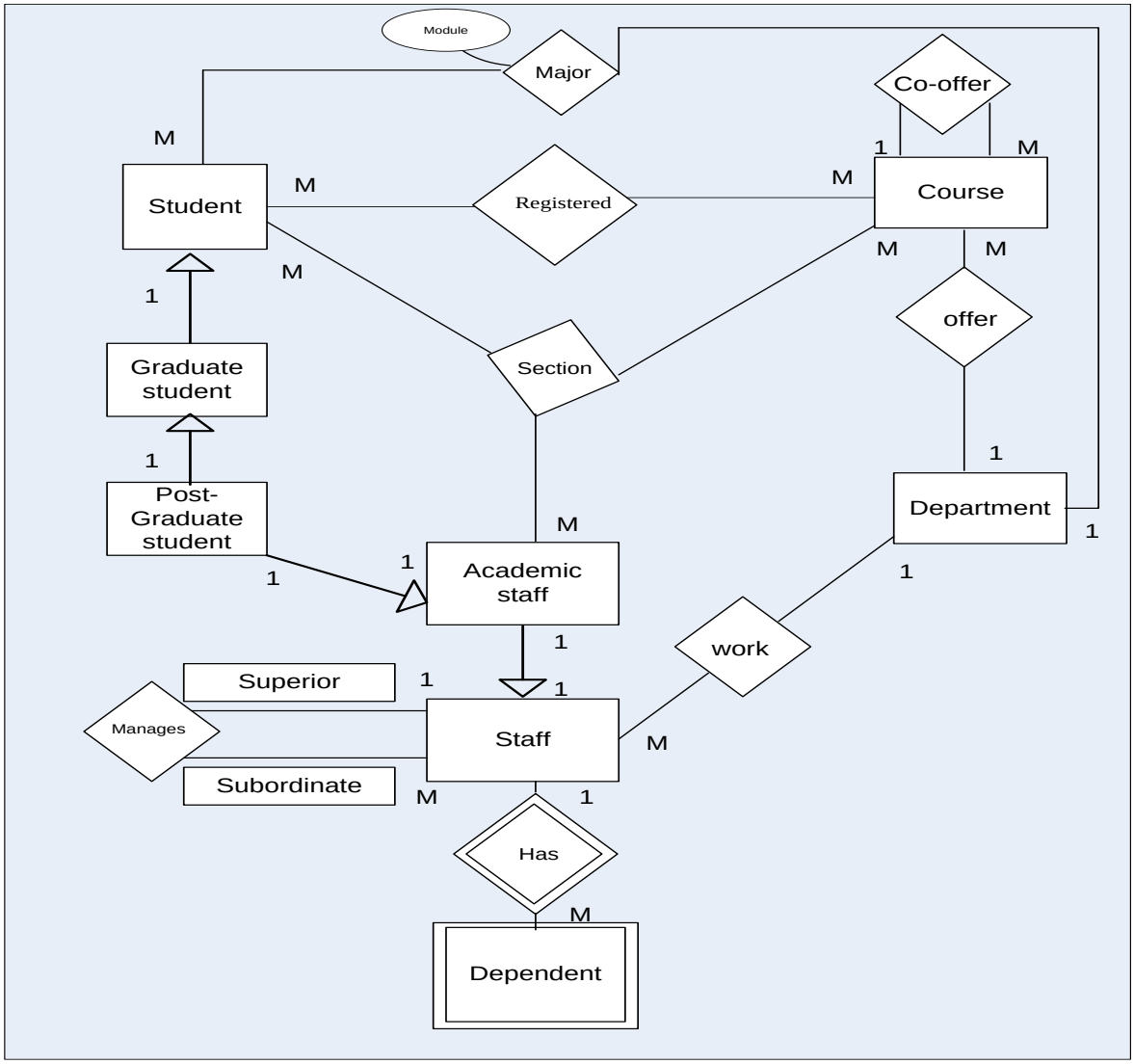


Figure 8-1: EER Diagram of the University Database

8.2.3 University relational schema

After generating the EER representing the database requirements, we used the algorithm as explained in Chapter Two, to convert the EER model to RM schema. The results are represented in Table 8.1 below.

Table 8-1: The University Database Schema

Tables	Primary key	Foreign key
Department(dept_id, dept_name, dept_phone),	dept_id	NA
Staff(staff_id, staff_ family_name, n-id, dept_id , manager_id,)	staff_id	dept_id (department) staff_id(staff-details) manager_id(staff)
staff-details(staff_id ,staff_first_name staff_mid_name, DOB, address, email, homephone, university extension phone)	staff_id	staff_id (staff)
academic-staff(staff_id, Post-held, specialty)	staff_id	staff_id (staff)
student(st_id, st_name, sex, module-name, dept_id)	st_id	dept_id (department)
graduate-student(st_id, research_area)	st_id	st_id (student)
post- graduate (st_id, staff_id, project_ group)	st_id	st_id (graduate-student) staff_id (staff)

Tables	Primary key	Foreign key
hobby(st_id , hobby_name)	(st_id, hobby_name)	st_id (student)
Dependent(d_id , staff_id dependent_name, relationship)	(d_id, staff_id)	staff_id (staff)
course (c_id, course_name, dept_id, course credit hour, co-offer)	c_id	dept_id(department) co-offer (course)
Registered(st_id, c_id)	(st_id, c_id)	st_id(student), c_id (course)
Section (st_id, c_id, staff_id)	(st_id, c_id, staff_id)	st_id(student), c_id(course), staff_id(staff)

Here are some design decisions made by the database designer:

- Consider the dependent entity as a weak entity.
- Dependent is a weak entity. Its primary key will have the owner entity (staff) primary key added to the discriminator which is dependent-id.
- We exclude the attribute grade, year and semester from the relationship registered temporary. We will add them on later to show the effectiveness of relationship attribute in ontology design
- The important information about staff is to be placed in one table and the remaining in another table.
- Determine that the Id number given by the university is the primary key for entities that has an alternative key. For instance the national number (n-id) in both academic staff and graduate student is a candidate key; even if the national number is restricted by not null and unique, the university id has meaning.
- The hobby is a multivalued attribute which needs to have a separate table.

8.2.4 SQL statement for University database

Our example in Appendix B, therefore, shows the corresponding SQL statements for the University database schema.

8.3 Ontology generation:

The generation of OWL ontology passed through many steps. Firstly the most important step was producing a unique identifier for the ontology and its definitions (classes and properties). Subsequent class creation steps would include applying the fragmentation, hierarchy, multivalued and default rules with their simultaneous datatype rules. Object properties would take place as a final step which would include the (Many-To-Many) relationship, self- relationship and default rules.

8.3.1 Producing Unique Identifiers (URIs) and Labels

Before discussing applying the transformation rules for our University example, we need to clarify the different role of names in databases and ontologies. Databases use the unique names of tables to identify them, whereas ontologies use a different concept in producing names and identifiers for ontologies constructs, including classes and properties.

The unique identifiers concept is considerable in OWL ontologies design. Therefore, each ontology construct, including classes and properties, should be identified uniquely. There are two methods to label concepts in ontologies: the first is using the names and attributes of tables, and names to label the corresponding classes and properties respectively. However, while database tables names are unique, the attribute of different tables can share the same name. Also, since ontology would not accept duplication names, the domain expert is responsible for revising ontology construct names, and for machine readability and semantic interoperability, the ontology elements' names should be given qualified names. The second possible solution is producing random names.

Because of the need for domain expert participation in choosing fully qualified names, our approach used only the names suggested from relational schema, and when

required, appending a database construct name with an integer in order to gain uniqueness (e.g. ID1, ID2 etc.).

Supposing the st-id is representing the student identification number in the Student table and st-id is the staff identification number in the Staff table; each one of these two attributes would be referring to different attribute meanings in the database design. However in ontology design they must have different name spaces.

8.3.2 Applying our rules on University example

We applied our heuristic rules for transforming database schema-to-ontology automatically.

8.3.2.1 Fragmentation rules

Before applying this rule, the user had to decide which solution was preferable as explained in Chapter 6. There are three solutions for fragmentation:

1- The database is correct and complete specified.
2- Using the row record method.
3- Consider it as a hierarchy.

In our example we chose the first case which was the default. Therefore:

- **Rule C_1: Fragmentation class rule:**

The fragmentation rule had two conditions. The first condition represented two tables sharing same primary key. The second condition represented the two tables' primary keys referring to each other table. Here we must indicate that the two primary keys are equal which means the condition does not accept the tables that have partial primary keys. The following tables are those that satisfied the first condition:

Staff-details → Staff,
Academic-staff → Staff,

Graduate-student → student,

Post -Graduate → Graduate-student

To eliminate all tables, which did not represent fragmentation tables, we applied the second condition and the result would be thus:

Staff-details → Staff

Therefore after applying the fragmentation rule to the University example the result was the creation of one class (Staff) for Staff and Staff-details tables.

- **Rule DP_1: Fragmentation datatype property rule**

The rule states that for all tables merged in one class create datatype property for each attribute. Not forming a foreign key except for those attributes which are part of the primary keys of the two tables and should therefore appear only once. For example the primary keys of the two tables Staff, Staff-details were represented by one datatype property. This datatype property subsequently used class Staff as a domain, since we integrated the Staff table and Staff-details table into Staff class. The following attributes would also have class Staff as their domain:

Staff (staff_id, n_id, staff_ family_name, staff_first_name staff_mid_name , DOB, address, email, home phone, university extension phone).

Noticeably dept_id, manager_id which are the foreign attributes would not have a datatype property corresponding to them.

8.3.2.2 Hierarchy rules

The hierarchy rule takes care of the inheritance between two tables or multiple inheritances for more than two parent tables.

- **Rule C_2: Hierarchy class rule**

The first condition is two tables sharing the same primary key. After applying this condition we achieve the same fragmentation results which are:

Staff-details → Staff, excluded by (fragmentation rule)

Academic-staff → Staff,

Graduate-student → student

Post -Graduate → Graduate-student

Here our tool already excludes all relationships applied for the fragmentation rule and the tables that have partial part of the primary key such as:

Dependent → Staff,

Hobby → Student,

Registered → Student,

Registered→ course,

Section → Student,

Section → course,

Section → academic-staff,

The second condition, which is one of the tables primary keys will refer to the primary key of the other table and this condition is applicable for:

Academic-staff → Staff,

Graduate-student → student,

Post -Graduate → Graduate-student

Here the creation for classes does not exist, such as Academic-staff, Student, Graduate-student and Post-Graduate. The system then maintained for each (IS-A) relationship the term subclass between the parent child classes in the ontology. The master class is Staff and the slave is Academic-staff. Therefore Academic-staff is a subclass of the Staff class. Similarly the Graduate-student will be subclasses of Student class; and Post-Graduate subclasses of Graduate-student.

The second case related to when the foreign key was disjoint from the primary key. Here we considered the inheritance existing of an implicit (1 -1) relationship between the two tables. This is true if and only when the foreign key has both UNIQUE and NOT NULL characteristics. From our example we had the two tables Staff and Post-Graduate where the Post-Graduate table holding the relationship between the foreign key was disjoint from the primary key. It is therefore obvious that the Post-Graduate is an Academic-Staff from database analysis. However the primary key of table Post-Graduate is not sharing the primary key of the Staff table. Although the foreign key (Staff-id attribute) in the Post-Graduate table represent an implicit inheritance since it is UNIQUE and NOT NULL. Therefore we defined the Post-Graduate class as subclass of the Academic-Staff class.

vii. Rule DP_2: Hierarchy datatype property rule

This rule played an important role insuring that each attribute would be assigned its according child or parent class, i.e. each class would have its own datatypes. The classes created by this rule were: Academic-staff, Student, Graduate-Student and Post-Graduate; whereas the Staff class already existed from the previous rule. In addition the attributes of class Staff were already assigned from the fragmentation datatype property rule.

The class Academic-staff will be the domain for the attributes Post-Held, Specialty which do not form a foreign key. The class Student will be considered as a domain for the attributes St_Id, St_Name, Sex and Module-Name and do not form a foreign key either.

Since the class Graduate-student is a subclass from Student class it will inherit all its attributes. The Post- Graduate class will inherit all the datatype properties of both Student class and Graduate-student class. Here the datatype property Project-Group assigned to the Post- Graduate class as a domain.

One of the distinctive ontological features is that child class inherit all the attributes of the parent class automatically; whereas the database enforces the designer to repeat the attributes manually.

Academic-staff (all Staff datatype properties, plus Post-Held, Specialty

Student (St_Id, St_Name, Sex, Module-Name)

Graduate-student (all student datatype properties and Research _Area)

Post -Graduate (all student datatype properties, all Academic-staff datatype properties and Graduate-student, Project-Group)

OWL classes 'overlap' until they have been stated to be disjoint from each other. Thus the post-graduate class can have two parent classes and will consider being a specialisation lattice with the subclass of (Graduate-student, Academic-Staff). As long as the disjoint axioms are not created between Academic-staff class and Student class, the reasoner would not detect inconsistencies in the ontology. The integrity of the ontology could be checked by the reasoner to ensure that our ontology was built correctly. In addition the class Post –Graduate shows the hierarchy inheritance since it subclass of Graduate-student and Graduate-student is also subclass of Student class.

8.3.2.3 Multi-valued rules

The multivalued attribute in a well-designed database will have its own table. The relationship between the owner table and multivalued table will have the cardinality ratio of (One-To-Many).

- **Rule C_3: Multi-valued class rule:**

There are two conditions that distinguish the multivalued relationship from other relationships. Firstly the primary key of the multivalued table is a range of composite attributes that will satisfy the relationship between tables below:

Dependent → Staff
Hobby → Student,
Registered → Student,
Registered→ Course,
Section → Student,
Section → Course,
Section → Academic-Staff,

The second condition is the part the primary key of the table is playing and the other part of the primary key is only one attribute which is not participating in inclusion dependency. All relationships representing (Many-To-Many), weak entities or *n*-ary will be eliminated by the second condition. After that the only table satisfying our rule is the relationship below:

Hobby → Student,

Since the class of student already exists no actions are executed.

- **Rule DP_3: Multi-valued datatype property rule**

Since the two tables are merged in one class, all the attributes will use the class created from the previous step as their domain. For our example all the student attributes were already allocated to the class Student, and for the Hobby table the attribute Hobby-name forming a multivalued attribute will also be allocated to class Student. Noticeably this

attribute could not have the feature of *Functional* property since it definitely had more than individual.

Student (all student datatype properties, Hobby-name)

8.3.2.4 Default rules

- **Rule C_4: Default class rule**

This step ensured all the remaining tables that did not participate in the previous rules (fragmentation, hierarchy, multivalued) would be examined by the default rule condition. The condition is creating classes for all remaining tables except the table representing the (Many-To-Many) relationship without attributes. In our example the remaining tables which did not satisfy the previous rules were:

Department, Dependent, Course, Registered, and Section

From such tables the only one to represent the (Many-To-Many) relationship was the Registered table. Therefore the tool would create classes for strong and weak entity; and the ternary (higher) relationship not involved in other rules.

Department, Course → strong entities, Dependent → weak entity, Section →ternary relationship

The default rule would take care of the following two cases.

- ***Case 1: Binary relationships with additional attributes***

If we slightly modify the specifications of the table Registered to include additional attributes such as semester and grade the table would subsequently form a class according to the default rule.

- ***Case 2: N-ary relationships***

For those tables that represented the ternary or higher relationship the default rule took care of creating corresponding classes on the ontology side.

- **Rule DP_4: Default datatype property rule**

The rule was concerned about assigning all attributes from the original table to its corresponding datatype properties, except the foreign keys. For the University example the classes created by the default rule would have the following datatype properties:

Department→ dept_id, dept_name, dept_phone
Course → c_id, course_name, dept_id, course-credit-hour
Dependent → d_id , staff_id ,dependent_name, relationship
Section → c_id, staff_id, st_id

In addition, the tables of the University database which did not form a class were:

Staff-details → merge with Staff class
Registered → many –to-many relationship
Hobby → merges with Student class

At the end of this phase the need for the domain expert participation was important. All datatype properties of classes therefore needed to be given a proper name before proceeding.

Finally, classes were created from Table 8.1 by class creation rules:

- Fragmentation rule→ (Staff+ Staff-details) = Staff
- Hierarchy rule→ (Academic-Staff, Staff), (Graduate-student, Student) (Post-Graduate, Graduate-Student), (Post- Graduate, Academic- Staff)
- Multivalued rule→(Hobby+ Student) = Student
- Default rule→ Department, Course, Dependent, Section

The corresponding code below represents the header of the ontology which includes the name space for OWL, RDF, RDFS and XML. However the header varied from application to application and hence we adopted the one able to work with Protégé – OWL.

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [

  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >

  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >

  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >

  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >

]>

<rdf:RDF xmlns="http://www.owl-ontologies.com/Ontology1290365199.owl#"

   xml:base="http://www.owl-ontologies.com/Ontology1290365199.owl"

   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

   xmlns:owl="http://www.w3.org/2002/07/owl#">
```

The following script represents the class creation (first stage). OWL segments for class creation are:

```
<owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="Academic_staff">

    <rdfs:subClassOf rdf:resource="#staff"/>

  </owl:Class>

  <owl:Class rdf:ID="course"/>

  <owl:Class rdf:ID="department"/>
```

```
  <owl:Class rdf:ID="Dependent"/>
  <owl:Class rdf:ID="Graduate_Student">
    <rdfs:subClassOf rdf:resource="#Student"/>
  </owl:Class>
  <owl:Class rdf:ID="Post_Graduate">
    <rdfs:subClassOf rdf:resource="#Student"/>
    <rdfs:subClassOf rdf:resource="#Academic_staff"/>
  </owl:Class>
  <owl:Class rdf:ID="staff"/>
  <owl:Class rdf:ID="Student"/>
  <owl:Class rdf:ID="Section"/>
</rdf:RDF>
```

For attribute representation we selected samples from University ontology. The first sample was the definition for the datatype property staff_family_name and had the class Staff as its domain; and the range was xml datatype property equivalent to database attribute type. The other samples introduced in the next section.

```
<owl:DatatypeProperty rdf:ID="staff__family_name">
    <rdfs:domain rdf:resource="#Staff"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
```

For full ontology classes and datatype properties creation refer to Appendix C.

One approach [41] has mixed ontology and database design. For example, if there were two classes represented in a hierarchical structure, the subclass would inherit all the datatype properties present in the superclass, so there would be no need to define the datatype property domain from the union of the superclass and the subclass.

For instance the Staff_id datatype presented in both Staff and Academic-Staff classes did not mean that we aggregated the two attributes by the collection term. However if we preferred to link the relationship between the weak and strong entity we could use the OWL term - Collection - since this relationship was tighter than the normal (One–To-Many). For example the relationship between the strong entity Staff and the weak entity Dependent can have the following code.

```
<owl:DatatypeProperty rdf:ID="staff_id">
  <rdfs:domain>
   <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Staff"/>
      <owl:Class rdf:about="# Dependent "/>
            <owl:Class rdf:about="# staff_id "/>
     </owl:unionOf>
   </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="&xsd;int"/>
 </owl:DatatypeProperty>
```

From the example Staff_Id this is a datatype property of the class Staff. Therefore the Academic-Staff class would not need to define the staff_id again since the subclass term granted all datatype properties of the parent class to be inherited by the child class.

8.3.2.5 Applying other SQL aspects to datatype property rules

For all attributes that did not participate in foreign keys could be one of seven cases:

- Firstly, an attribute will be *Functional* if it is not representing a multivalued datatype property. In the University example the representation for the name of a student will be *Functional* since each student has only one name.

```
<owl:DatatypeProperty rdf:ID="Student_name">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="&xsd;string"/>
```

```
</owl:DatatypeProperty>
```

- The second case is the attribute standing for multivalued; and thenceforth this datatype property must not be restricted with *Functional* axiom. In University ontology, only the datatype property of student Hobby cannot be functional.
- The third case is if the attribute is a primary key or have the NOT NULL and UNIQUE constraints together; the corresponding datatype property will thus have the minimum cardinality and maximum cardinality of one. The attribute Staff_Id represent a primary key in the Staff table and therefore the datatype property Staff_Id of class Staff will have both minimum cardinality of one and maximum cardinality of one.

```
<owl:Restriction>
<owl:onProperty rdf:resource="# staff_id "/>
<owl:minCardinality>1</owl:minCardinality>
<owl:maxCardinality>1</owl:maxCardinalit>
</owl:Restriction>
```

For the case of minimum cardinality equal to maximum cardinality then cardinality restriction is enough for specification. So the previous code could easily be rewritten as:

```
owl:Restriction>
    <owl:onProperty rdf:resource="# staff_id "/>
    <owl:Cardinality>1</owl:Cardinality>
</owl:Restriction>
```

The same thing is applicable for the attribute which have NOT NULL and UNIQUE constraints, and will be restricted to cardinality of one. The national number of the staff will satisfy these conditions.

```
<owl:Restriction>
    <owl:onProperty rdf:resource="# national_id " />
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
</owl:Restriction>
```

- The fourth case is the attribute with Not Null constraint, the corresponding datatype property to this attribute will be assigned with minimum cardinality of one.

```
<owl:Restriction>
 <owl:onProperty rdf:resource="# staff_ family_name" />
 <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
</owl:Restriction>
```

- The fifth case if the attribute is holding UNIQUE constraint; thenceforth the corresponding datatype property will have an additional cardinality restriction on that property with maximum cardinality of one.

```
<owl:Restriction>
 <owl:onProperty rdf:resource="# module-name" />
 <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
</owl:Restriction>
```

- The sixth case for representing the database attribute characteristic is the check in and thus can be represented by an enumerated datatype property. The attribute Post-Held in the Academic-Staff table is an example of enumerated datatype property in University ontology. The following is the OWL matching segment for the options (‘Teacher_Assistance’,‘Instructor’,‘Assistant_professor’,‘Associate_professor’ and ‘Professor’).

```
<owl:DatatypeProperty rdf:ID="Post-hold">
  <rdfs:domain rdf:resource="#Academic-Staff"/>
  <rdfs:range>
   <owl:DataRange>
    <owl:oneOf>
     <rdf:List>
      <rdf:firstrdf:datatype="&xsd;string"> Teacher Assistance</rdf:first>
      <rdf:rest>
       <rdf:List>
        <rdf:first rdf:datatype="&xsd;string">Instructor</rdf:first>
        <rdf:rest>
```

```
            <rdf:List>
            <rdf:first rdf:datatype="&xsd;string">Assistant_professor</rdf:first>
              <rdf:rest>
                <rdf:List>
                  <rdf:first rdf:datatype="&xsd;string"
                    >Associate_professor</rdf:first>
                  <rdf:rest>
                    <rdf:List>
                    <rdf:first       rdf:datatype="&xsd;string">Professor</rdf:first>
                      <rdf:rest rdf:resource="&rdf;nil"/>
                    </rdf:List>
                  </rdf:rest>
                </rdf:List>
              </rdf:rest>
            </rdf:List>
          </rdf:rest>
        </rdf:List>
      </owl:oneOf>
    </owl:DataRange>
</rdfs:range>
```

- The last case for representing the database attribute is the SQL term default. The corresponding datatype attribute will have the *Owl:hasValue* term. The following Owl syntax shows the example of the relationship of class Dependent.

```
<owl:Restriction>
 <owl:onProperty rdf:resource="#Relationship" />
 <owl:hasValue rdf:resource="#Parent" />
</owl:Restriction>
```

The Figure 8.2 below has summarised all SQL characteristics applicable to database attributes and their equivalent ontological constraints.

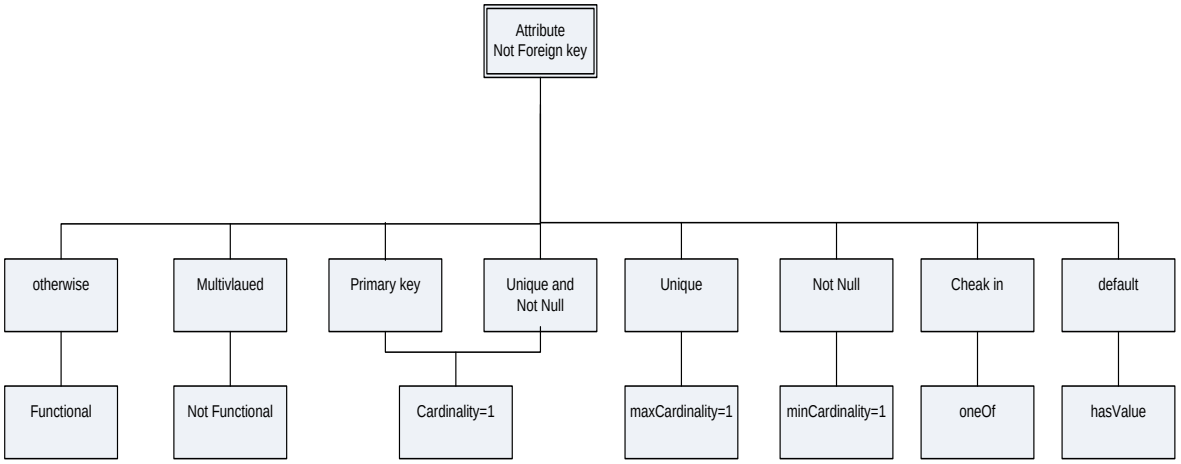


Figure 8-2: SQL and OWL Equivalent

8.3.3 Rules for the creation of object properties

The foreign keys in database schema played the role of building the relationships between tables. Since some class rules take care of some foreign keys cases. Therefore only the foreign keys that do not participate in any of class rules (fragmentation, hierarchy, or multivalued) can be checked by the following object property rules.

The foreign keys, for the university database, which participated in class creation, have been defined in Table 8.2. These did not require an object property to represent them.

Table 8-2: Foreign keys participate in University class creation

table	Foreign key	Reference table	Situation
Staff- details	Staff_id	Staff	Staff- details integrate into Staff
Staff	Staff_id	Staff- details	Staff- details integrate into Staff
Academic- Staff	Staff_id	Staff	Academic- Staff subclass Staff
Graduate-student	St_id	Student	Graduate-student subclass Student
Post- Graduate	St_id	Graduate-Student	Post- Graduate subclass Graduate-Student
Post- Graduate	Staff_id	Academic-Staff	Post- Graduate subclass Academic- Staff
Hobby	St_id	Student	Hobby integrate into Student

8.3.3.1 Rule OP1: Object property rules for binary relationships (many-to-many)

The rule is creating a pair of inverse object properties between the two classes corresponding to the two tables involving in Many-To-Many relationship. In the University example the Registered table represents the (Many-To-Many) relationship, not represented by a class. Therefore we can create (student-registered) object property with class Student as its domain and class Course as its range. In addition we created another object property with name (course-available); Course class domain, and the range of Student class. The two object properties are inverse of each other. The object properties of this case cannot be functional since each student can register on more than one course and each course can have more than one student. The following is the OWL segment representing each class object properties.

For **Course Class**:

```
<owl:ObjectProperty rdf:ID="course_available">
  <rdfs:domain rdf:resource="#course"/>
  <rdfs:range rdf:resource="#Student"/>
  <owl:inverseOf rdf:resource="#student_registered"/>
</owl:ObjectProperty>
```

For **Student Class**:

```
<owl:ObjectProperty rdf:ID="student_registered">
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#course"/>
  <owl:inverseOf rdf:resource="#course_available"/>
</owl:ObjectProperty>
```

8.3.3.2 Rule OP2: Object property rules for a relation with unary relationship (self-relation)

The self-relationship is represented by two inverse object properties in the ontological side. The relationship of Manager is an example for this matter. In University ontology the object property (has-superior) will be *Functional* with Staff class domain and range. The inverse to it is (has-subordinate) with same domain and range class, although without assigning the *Functional* characteristic to it.

The corresponding OWL descriptions are as follows:

```
<owl:ObjectProperty rdf:ID="has_subordinate">
   <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
   <rdfs:domain rdf:resource="#staff"/>
   <rdfs:range rdf:resource="#staff"/>
   <owl:inverseOf rdf:resource="#has_superior"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="has_superior">
   <rdf:type rdf:resource="&owl;FunctionalProperty"/>
   <rdfs:domain rdf:resource="#staff"/>
   <rdfs:range rdf:resource="#staff"/>
   <owl:inverseOf rdf:resource="#has_subordinate"/>
 </owl:ObjectProperty>
```

8.3.3.3 Rule OP3: Object property default rule

The default object property rule takes care of all foreign keys that are not participates in class creation nor participates in the two former object property rules. The default rule will handle the foreign keys of the One-To-Many relationship; the *n*-ary relationship foreign keys and the Many-To-Many relationship with additional attributes as well and the strong –weak entity relationship.

From the Table 8.3 below, which contained the foreign keys not participates in class creation, we noticed that Registered foreign keys are handled by Rule-OP1 Manager_Id of Staff table; and Co-offer of course are handled by the Rule-OP2. Whereas, the remaining foreign keys, representing different relationships between tables, would be handled by the default object rule.

Table 8-3: Foreign keys participate in object properties

Table	Foreign key	Reference table	Relationship name
Staff	Dept_id	Department	Work for
Staff	Manager_id	Staff	Managed by
Student	Dept_id	Department	Major
Course	Dept_id	Department	Offer

Table	Foreign key	Reference table	Relationship name
Course	Co-offer	course	Co-offer
Dependent	Staff_id	Staff	Has Staff
Registered	St_id	Student	Student Registered
Registered	C_id	course	course Registered
Section	St_id	Student	Student Section
Section	C_id	course	course Section
Section	Staff_id	Academic-staff	Academic-staff Section

We noticed that all foreign keys created by the tool did not have proper names. For example the foreign keys of table Registered had the names Student-Registered-course and course-Registered-Student which included the domain and the range with the relationship. Therefore we modified the names to be rational.

The default rule would handle the foreign keys of tables Staff, Student, and Course and referred to table Department by two inverse object properties. For example the relationship work between tables Staff and Department would have the following OWL description for the corresponding classes.

```
<owl:ObjectProperty rdf:ID=" work_in">
        <rdfs:domain rdf:resource="# Staff "/>
      <owl:inverseOf rdf:resource="# has _worker "/>
        <rdfs:range rdf:resource="# department "/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID=" has _worker">
      <rdfs:domain rdf:resource="# department "/>
      <owl:inverseOf rdf:resource="# work_in"/>
      <rdfs:range rdf:resource="# Staff "/>
  </owl:ObjectProperty>
```

The description above is therefore demonstrating the relationship as Staff *work in* Department and Department *has worker* Staff. The foreign key between the strong entity table Staff; and the weak entity table Dependent would have the same treatment of the One-To-Many relationship.

The third case considered the (Many-To- Many) binary relationship with additional attributes and higher degree relationships. Since in this case the class for the relationship existed and subsequently each foreign key will have two object properties to represent it.

From the University database, Section table is a ternary relationship, therefore the foreign key referring to Student table will be represented by two inverse object properties (is- taught-by, has-student). For the foreign key referring to Staff table, the two inverse object properties (has-staff, staff-Section) will demonstrate it. Finally the foreign key referring to Course table will have the two inverse object properties (has-course, course-taught-by); and for all higher degree (n>2) relationships represented by class will have n* (n-1) object properties.

8.3.3.4 Applying other SQL aspects to object property rule

Object properties identified from foreign keys and their characteristics could be obtained from various combinations of uniqueness and null restrictions. The default rule took care of object properties creation. Our tool simultaneously applied the algorithm of object property restrictions obtained from SQL aspect. There were two categories of foreign keys.

The first category related to when the foreign key was disjoint from the primary key of its table. Here we had three cases. The first case, if the foreign key is null and not unique, and therefore the corresponding object property will be functional and its inverse will have minimum cardinality of 0. An example of that is the Offer relationship between Course and Department. The second case, if the foreign key is not null and not unique; and subsequently the corresponding object property will set its cardinality to one and set its inverse minimum cardinality to 0. The Major relationship between the Student and Department table has been demonstrated by the syntax below.

For Student Class:

```
</owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Major">
    <rdfs:domain rdf:resource="#Student"/>
```

```
    <rdfs:range rdf:resource="#department"/>
    <owl:inverseOf rdf:resource="#invMajor"/>
</owl:ObjectProperty>
<owl:Restriction>
    <owl:onProperty rdf:resource="#Major"/>
    <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
</owl:Restriction>
```

For Department class:

```
</owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="invMajor">
    <rdfs:domain rdf:resource="#department"/>
    <rdfs:range rdf:resource="#Student"/>
    <owl:inverseOf rdf:resource="#Major"/>
</owl:ObjectProperty>
<owl:Restriction>
    <owl:onProperty rdf:resource="#invMajor"/>
    <owl:minCardinality rdf:datatype="&xsd;int">0</owl:minCardinality>
</owl:Restriction>
```

The third case shows whether the foreign key is null and unique; and subsequently the relationship for (one-to-one) will be represented by two *Functional* object properties inverse from each other. The last case when the foreign key is not null and unique, at this point the hierarchy rule will take care of it.

The second category when the foreign key is part of the primary key, such as in the tables Section and Dependent. Here the corresponding object property and its inverse, both will set their cardinality to one. The syntax below shows the Section class object properties:

```
<owl:Class rdf:ID="Section">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty rdf:resource="#has_course_Section"/>
```

```
        <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#has_student_Section"/>
        <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#has_staff_Section"/>
        <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

Since the primary key is restricted to being not null and unique therefore the question here is if the foreign key is part of the primary key this makes it eligible to inherit the primary key characteristics. However clarification for this issue is the primary key of the relationship table as a concatenation of the foreign keys. . Therefore the restriction of uniqueness would not be applicable for each foreign key independently. However it is applicable for the entire component of the primary key. This is the reason behind not assigning the two object properties representing the foreign key the functor of Functional which represent the one-to-one relationship.

As for the Not Null restriction it was applied for each foreign key which let us set the cardinality of one to the corresponding object properties. We could not force the combination of all object properties part of *n*-ary class to be UNIQUE and NOT NULL, since OWL did not support restrictions on combination. Therefore we could use an alternative way to express such constraints. For example use of SWRL language or any other language rules to express these facts.

8.4 Implementation

Our transformation rules approach was applied on a prototype tool. This tool considers automatic transformation of relational databases to ontologies, where the quality of

transformation is also considered. The prototype has been implemented in the tool named SQL2OWL. This tool can be applied to any relational database management system that supports SQL, because the tool does not rely on any SQL dialect. The function of the tool has been to find a best candidate for each relational database construct and its matching in ontology. This excludes any construct that does not have an equivalent match in the ontology side, such as triggers. More precisely this tool has the capability of automatic transformation of a relational database (written in SQL) to an ontology (written in OWL). The complete set of our transformation rules was arranged stepwise, and that enables our approach to be executed directly by our prototype.

As its core, the tool is considered a transformation engine that parses an SQL script and generates an OWL file; that construct an ontology, including definitions (classes, properties and restrictions) and instances (values and individuals). The tool required minimum user interaction as participation included only selecting or specifying the name of an SQL script file and the name for an OWL file.

- Prototype structure:

The tool implementation was based on Java 2 v1.4.2 platform. We parse the database schema file and stored the schema data as Java ArrayList classes.

Our tool was functional and flexible as we considered these characteristics during the implementation process. In addition the prototype was user friendly with an intuitive interface (graphical user interface), also easy to use, as shown in Figure 8.3. The prototype consisted of six parts:

- Part 1: menu bar,
- Part 2: loading SQL statement,
- Part 3: database analysis,
- Part 4: the rule options part,
- Part 5: rules activities,
- Part 6: result ontology.

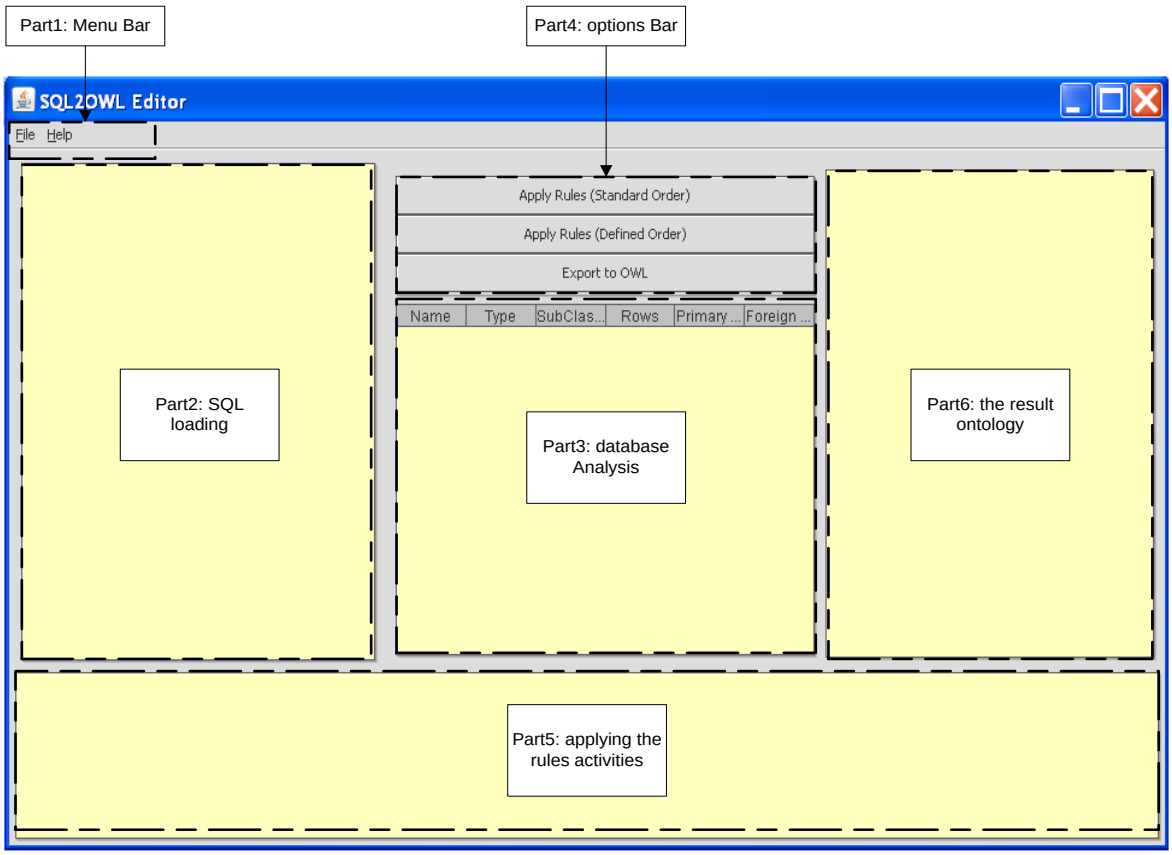


Figure 8-3: Prototype snapshot parts

The following section will explain the functions of each part.

8.4.1 Part 1: menu bar

This part allowed the user to specify the SQL file as a source, and to save the produced OWL ontology as target.

8.4.2 Part 2: Load part

This part was responsible for showing the loaded SQL file script. It also highlighted each command of SQL statements with bold text. In the case that the SQL file was not available we **dump** the source database in order to obtain the SQL statements which included:

- CREATE table commands.

- INSERT records into tables.

The create commands described the relational schema of database, whereas the insert command represented the last stored data records for each table.

8.4.3 Part 3: database analysis

After loading the database statement, the tool would subsequently parse the SQL statements, to group each command part together in order to analyse the database. The tool could begin to analyse the database in order to capture the following components:

- Name

  The Name of relation will be considered as the name of class in the ontology. Here the column The Name of Part 3 specifies every relation in the database tables, even tables which represent relationships or multivalued attributes.

- Type

  This column will classify tables to either class or relation. Those tables which can result in a class or integrate into other class will have a class type. As for binary relationship tables without additional attributes, they will be assigned relation as a type which means this type of relations would not convert to a class.

- Subclass of

  This will consider the tables which are sharing the same primary as subclasses unless the fragmentation rule is applicable.

- Rows

  Rows indicate the number of rows held by a relation. This can be done by counting the number of insert statements of each table.

- Primary keys

All the attributes form the primary key of each relation.

- Foreign keys

  All attributes participate as foreign keys of each relation.

Each component would be facilitated in one of the creation rules. For example Name of relation would be assigned to be the name of classes in the ontology. The primary keys and foreign keys were utilised in verifying that exactly one rule was applicable on each relation, i.e. which means there are no two rules applicable on one relation; and there is no relation without a valid rule. The foreign keys will also be facilitated in creating object properties.

8.4.4 Part 4: Options part

This part included three push buttons options:

i. Apply rules (standard order)

   This option would follow the approach rules order which meant applying class and datatype creation rules as our approach suggested by the order of the fragmentation rule; thenceforth the hierarchy, the multivalued rule and eventually the default rule. Following on from this, the tool started to build object properties axioms, starting with Many-To-Many relationships, then unary relationships and finally the rest of the relationships.

   Here the order was important and manipulating of the order may have resulted in an incorrect outcome. In addition some rules could not be applied before another. For instance the hierarchy rule could not be applied prior to the fragmentation rule.

ii. Apply rules (defined order)

   This choice allowed the user to try each rule independently to determine its outcome. Another purpose of this option was to give the user the ability to omit the rule of fragmentation if it was considered out of database interest. This option also signified the flexibility of our tool.

iii. Export to OWL

The third option was to produce the final Owl ontology in the exchange syntax.

Discussion:

We considered the order suggested by our approach, since following the order procedure in applying the approach rules would enhance the tool performance; and thus reduce the time of generating the target ontology by narrowing the set of tables applied for each rule, especially for large scale databases. Therefore the procedure of applying the transformation rule contained four steps:

a. Identifying tables of Many-To-Many relationship without an additional attribute which would not have a corresponding class.
b. Identifying tables sharing the same primary key. Therefore other tables that do not satisfy this condition will be eliminated from the fragmentation and hierarchy rule. The system subsequently applies the fragmentation rule if it is applicable; otherwise applying the hierarchy rule in sequence.
c. Identifying tables in which part of their primary key refer to other tables. Thus weak entity and multivalued tables will be nominated and the *n*-ary relationship will be eliminated. The tool applies the multivalued attribute rule afterward.
d. The remaining tables will create corresponding classes.

The procedures of other approaches [36, 41, 45, and 46] for applying their class creation rules have only considered eliminating the binary relationship of Many-To-Many cardinality. Therefore they would only have one rule to represent class creation and one rule for class hierarchy. Omitting fragmentation problem and excluding multivalued tables from their rules would enhance the runtime; however it would also affect the result ontology. This has shown therefore the functionality of our rules which have been able to cover the space of all possible relations cases.

Another distinguished feature of our system was the procedure of applying the fragmentation rule as below.

The three options whilst applying the fragmentation rule are outlined:

a. Complete and correct specification database where each table refers to the other.

b. Using the count method for database data.

c. Treat fragmentation as hierarchical.

Our system applied these three options in sequence looking for tables which referred to each other. If not available, the count method to calculate the record of each table would be used. If the first option was not applicable and data not offered then all one-to-one relationships sharing the same primary key were to be treated as a hierarchy.

Any result could be modified after generating the OWL ontology.

8.4.5 Part 5: The activity part:

This shows the activity of applying each rule and its status and whether it was applicable or not.

8.4.6 Part 6: The ontology produced

This part showed a complete OWL ontology including all the definition of classes and properties and axioms. Figure 8.4 has shown a snapshot of the tool after applying the rules and receiving the result.

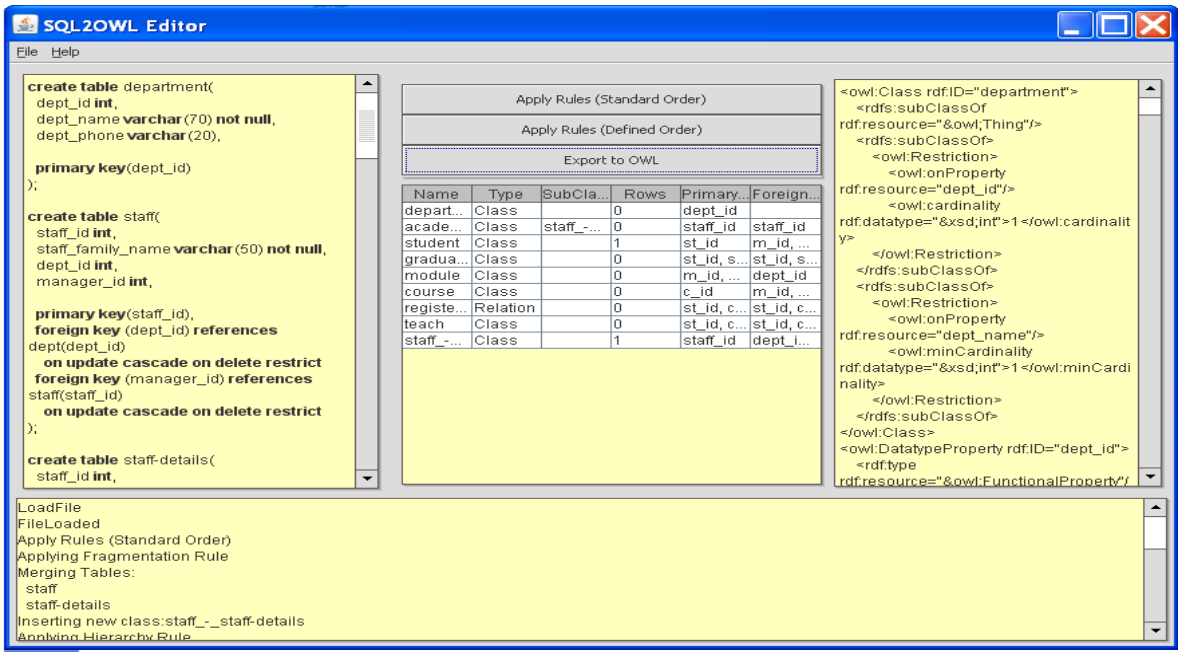


Figure 8-4: Prototype snapshot

8.5 Summary

This chapter focuses on two parts. The case study is the first part which includes specifying a University database as an example. This specification process goes through three steps. The first step is the conceptual model depicted in the EER diagram. Then the second step is to produce the relational schema. After that the physical model produced by SQL-DDL language. The second phase is producing the ontology starting by class creation rules along with datatype creation rules then the object property rules.
In the second part we built a tool that applied our transformation rules. This tool accepts an SQL-DDL script then transforms it to OWL ontology language.

CHAPTER NINE: EVALUATION

Objectives

- Compare our approach and other transformation approaches.
- Define criteria for evaluation.
- Evaluate the results of our approach compared to a manual one.
- Show the extent of the success of our approach.

9.1 Introduction

This chapter will assess our approach with a multi-faceted evaluation. It begins with a general evaluation including a comparison between our approach and other potential approaches. The second technique applies a Score Scheme. After this the approach is tested against a variety of criteria including; quality of transformation, completeness, and scalability.

The evaluation process must consider all the disparities between database and ontological modelling presented in Chapter 5.

9.2 General Comparison

This stage of the assessment evaluates our approach by comparing it to other possible approaches. Table 9.1 shows the source, target and the technique for each approach, in addition to specifying formality. The approaches included in this comparison concentrate on the transformation of a database to ontology, where ontology does not already exist.

Table 9-1: Transformation Approaches General Characteristic

Approach	Source	Target	Transformation	Formal/ Informal
Stojanovic, et al.	SQL	F-logic	Automatic	Formal
Astrova, et al.	SQL	OWL-Full	Automatic	Informal
M. LI ,et al.	Relational schema	OWL-DL	Automatic	Informal
Upadhyaya *et al.(* ERONTO)	EER	OWL-Full	Semi- automatic	Informal
Sonia *et al.* (R2O)	Metadata + SQL	OWL-DL	Automatic	Informal
Tirmizi, et al.	SQL	OWL-DL	Automatic	Formal
Benslimane, et al.	HTML+SQL	OWL-DL	Semi- automatic	Informal
Our approach SQL2OWL(source SQL)	SQL alone	OWL-DL	Automatic	Formal
Our approach SQL2 OWL (source: SQL+EER)	SQL & EER + Metadata	OWL-DL	Semi- automatic	Formal

Some approaches defined their transformation rules with mapping rules. As has been previously stated, mapping requires the existence of both database and ontology. Therefore all these techniques are considered regarding transformation. The procedure applied here aims to maintain a general comparison between our approach (SQL2OWL) and different approaches through multiple design criteria, as shown in Table 9.2. The comparison given focuses on the strengths and weaknesses of each approach and its limitations.

Table 9-2: Transformation Approaches Rules Characteristic

Criteria \ Approach		Stojanovic, et al.	Astrova, et al.	LI ,et al.	Upadhyaya et al. ERONTO	Sonia et al. R2O	Tirmizi, et al.	Benslimane, et al.	SQL2OWL
Class	Fragmentation	User decide	Uncover	Mix with hierarchy	Not applicable	Not applicable	Excluded by assumption	Mix with hierarchy	☑
	hierarchy	☑	☑	Mix with Fragmentation	☑	☑	☑	Mix with Fragmentation	☑
	Multiple inheritance	☑	☑	☑	☑	☑	☑	☑	☑
	Multi-valued	Uncover	Uncover	Uncover	☑	☑	Uncover	Uncover	☑
	Binary (M-M)relationship with additional attribute	Uncover	☑	Uncover	☑	☑	☑	Uncover	☑
	Ternary and higher relationship	☑	☑	☑	☑	☑	☑	☑	☑
Object Properties	Binary (M-M)relationship	☑	☑	☑	☑	☑	☑	☑	☑
	Ternary and higher relationship	Partial solution	Partial solution	Partial solution	☑	Partial solution	Partial solution	Partial solution	☑
	Unary relationship	Uncover	Wrong solution	Uncover	Uncover	☑	☑	Uncover	☑
pe Propert	Functional	Uncover	☑	Partial solution	☑	☑	Wrong solution	Uncover	☑

Criteria \ Approach		Stojanovic, et al.	Astrova, et al.	LI ,et al.	Upadhyaya et al. ERONTO	Sonia et al. R2O	Tirmizi, et al.	Benslimane, et al.	SQL2OWL
	Cardinality	Uncover	☑	☑	☑	☑	☑	☑	☑
	Value restriction	Uncover	☑	Uncover	☑	☑	☑	Uncover	☑
	Enumerated property	Uncover	☑	Uncover	Not Possible	Uncover	☑	Uncover	☑
	Composite attribute	Uncover	Uncover	Uncover	☑	Uncover	Uncover	Uncover	☑ EER
	Transitive & symmetric	Not Possible	Not general solution	Not Possible	Not Possible	Not Possible	Not Possible	Not Possible	Not Possible

Table 9.2 shows a general comparison between different approaches which tried to transform relational model to ontological model using different database sources. We base our comparison on the three elements that built an ontology (classes, object properties and datatype properties). In the class creation phase, most approaches fail in distinguishing between fragmented tables and hierarchy tables, however, our approach success in solving this problem by obtaining a correct analysis which differentiate the specifications of each case. The second problem is that most of the approaches neglect the case of multi-valued tables while our approach considers it. The third problem is that many approaches do not consider the binary relationship having (Many-To-Many) cardinality ratio with additional attribute in their specifications. The approaches such as ERONTO and the R2O would not face the problems of fragmentation and multi-valued tables since they are using the conceptual model (EER) as their source.

In object property creation phase, most of the approaches decompose a higher degree relationship to binary relationships in order to represent the object properties. However they miss characteristics such as Inverse Functional and Functional which ensure that the relationships exist as a whole. In the case of the unary relationship, most of the approaches do not consider it. In datatype property creation phase, there are different

criteria to compare between the approaches. First of all, the Functional characteristic, since each field in database have only single value, then all datatype properties should be Functional. This assumption is not true for datatype obtained from multi-valued attribute. Many approaches facilitate the logical model only; however, logical model cannot obtain all the semantics in the conceptual model such as composite and complex attribute characteristics. Finally Astrova, et al. approach fails in the same aspect by using a general solution in representing the transitive and symmetric cases.

Table 9.3 provides an explanation of each approach in terms of its strength and its limitations. It is clear that the ERONTO approach has a lot of strengths since it obtains more semantic characteristics of the ontology from the EER source directly; thus, the problem is not with the approach itself, rather with the availability of source data. Therefore we cannot rely on this source alone, even though the results show more semantics than the RM. For this reason, in our approach we used the conceptual model for the validation process.

There are some criteria, which can affect the overall evaluation of each approach, that are important to consider besides the creation of the ontology components. One of these is availability and the other is the automation of the approach. These have the most significant influence on the evaluation process, in addition to the formality of the approach. There is no objection if the source is hard to obtain; the opportunity for constructing a new ontology from scratch is less complex. Therefore we give the above criteria a weight, which makes the evaluation more scalable.

Criteria are classified into three levels. First level is a very important criterion which can get the score of three. Availability of the source and the structure of the ontology classified from this category. The second level of criteria will gain the score of two. The automation is the only criterion considered from the second level. The last level with score of one contains the less important criterion or the supplementary criterion. Formality represents the less important criterion. Also, an additional source utilised by an approach can be considered as an additional criterion too. Now, we base our weighting on five criteria. And in order to get the total weight of 100 percentages, each criterion score weight will be multiply by 10. So the availability and structure of the

ontology will get 60 for both of them. And the automation would have the weight of 20. Finally, the weight of 10 will be granted to formality and the same weight to the additional source. For the weighting of the availability of each main source we can derive 30% for SQL-DDL or meta-data, since it is available by requesting a database. In the case of using an EER model as a source the approach the score is just 20% as it difficult to obtain. Using more than one source adds an extra 5% to the approach if the source is data or EER, and 10% extra will be given to an approach based on HTML, with the condition that the total weighting for availability does not exceed 40%. Notice that no single approach gets the full percentage for availability since all of them require some information to be attained by the database owner. For the automation weighting; if the approach was fully automated this would be 20%. If there is some a user intervention required (user decision) then there will be 5% exclusion, whereas semi-automatic approaches will receive 10%. For the ontology concepts we assign 2% for each component in table 9.3 with a total of 30%. Finally formal approaches will get score 10%.

Table 9-3: Transformation Approaches with Weighted Criteria

Criteria Weight / **Approach**	Availability 40%	Automation 20%	Ontology concepts 30%	Formality 10%	Total weight 100%
Stojanovic, *et al.*	30%	15%	12%	10%	67
Astrova, *et al.*	30%	20%	20%	0	70
LI ,*et al.*	30%	15%	14%	0	59
Upadhyaya, *et al.* ERONTO	20%	10%	24%	0	54
Sonia, *et al.* (R2O)	35%	20%	24%	0	79
Tirmizi, *et al.*	30%	20%	19%	10%	79
Benslimane, *et al.*	40%	10%	10%	0	60
SQL2 OWL(SQL source)	30%	20%	26%	10%	86
SQL2 OWL(EER &SQL)	35%	15%	28%	10%	88

- **Explanation:**

There are two kinds of sources that can represent a database: (i) loosely descriptive sources such as HTML and meta-data; (ii) tightly descriptive sources such as relational models written in SQL, EER.

The easiest source, which can be obtained without the need for database owner cooperation, is HTML. However, HTML creates links between forms of websites and the deep web database. Therefore we cannot count it as an actual database model, since it does not capture the hidden structure of the database. For case of data availability we consider retrieving all database records through a website to be an impossible task. Therefore data alone cannot represent the whole structure of a database. For the two actual database models we require database owner cooperation, which is available at different levels according to sources, as shown below:

- SQL alone;
- EER alone;
- Data alone;
- SQL with Data;
- SQL with EER;
- SQL, EER, and Data.

After dumping a database we can catch both SQL structure and data. However, most time data is confidential therefore the database administrator can eliminate this to avoid sharing it with other parties. Alternatively it may be shared after some precautions have been implemented, e.g. filtering some fields of tables in order to maintain security.

Theory: while the extend entity relational model is semantically richer than the RM model, the relational model written in SQL-DDL is the better source for transforming database to ontology for the following reasons:

- Availability.

- Accurate representation of the current database with the most recent modifications.
- It can produce the target ontology automatically with minimal user interaction.

The conceptual model (EER) can fail to guarantee availability for either one of these reasons:

- Missing model.
- Primarily not implemented during the database conception.

Additionally, it may not represent the current database as there is no documentation for any alteration to the database. These limitations with using EER sources lead us to the conclusion that; although obtaining ontology from a conceptual model might be better than that obtained from a relational model, it is not appropriate to rely on missing or unavailable sources.

Depending on three different sources gives our approach the ability to capture more semantics without a need for filtering the redundancy. Since our system relies on SQL source alone in order to produce the initial ontology, we can utilise the second source (if it exists) for the enriching and validation stage. The third source can then be used for filling in the ontology (knowledge base generation).

9.3 Experimental

9.3.1 Experimental Specification

The objective of this experiment is to evaluate the automatic approaches to producing ontologies in comparison to manual ones. This test will also show whether our heuristic rules can generate good quality ontologies or not.

The experimental phase includes many steps:

1- Choose database schema specification.
2- Generate a corresponding ontology by domain expert.
3- Generate automatically the corresponding ontology using three different approaches based on their SQL-DDL source.

4- Generate a corresponding ontology with our approach using SQL-DDL as the only source.
5- Assess the results of the four approaches compared to the domain expert ontology benchmark.

The manually created domain ontology is a non-trivial "benchmark", since it includes merging tables and creating classes to represent some concepts, such as composite attribute (a trivial transformation involves mapping from a single source table to a single target class). Also, we consider the manually-created domain ontologies as a "gold standard" to evaluate our approach result, compared with the other transformation approaches results.

We choose the approaches that totally depend on SQL-DDL as a source and can automatically produce ontology. Li *et al.*[41], Tirmizi *et al.* [45], and Astrova *et al.* [46] approaches are the three most comprehensive SQL-DDL approaches and their resulting ontology is automatically produced. We have avoided any approach that ignores the hierarchy rule. For instance Dogan and Islamaj's approach [37] which produces a flat ontology, since they disregard building inheritance their ontology looks relational. For our approach we use the automatic element, which totally depends on information obtained from the SQL source alone. This ensures the fairness of the evaluation process, since all participatory approaches use the same source applied without user interventions.

In this section the comparison between all the approaches participating in the evaluation for the same input (SQL-script) with determined specification is given. We use the example of the University database from Chapter 7. However, a Transcript table is added to represent the relationship of (Many-To-Many) cardinality with additional attribute Grade to University schema; as shown in Table 9.4. Thus our evaluation example is deemed comprehensive, since it includes all possible relationship cases.

Table 9-4: Table Transcript Added to University Database

Table	Primary key	Foreign key
Transcript (st_id, c_id,grade)	(st_id, c_id)	st_id(student),c_id (course)

9.3.2 Abstract Syntax (Normative)

The ontologies have been produced in abstract syntax for all four approaches participant in the evaluation phase, in addition to domain expert ontology. Figure 9.1 shows part of the domain expert ontology in abstract syntax, and the full version ontologies are demonstrated in Appendix C in exchange syntax.

```
Ontology(<urn:sql2owl>

Class(<Person> partial ...)

Class(<Department> partial ...)

Class(<Student > partial ...)

Class(<Staff > partial ...)

Class(<Dependent > partial ...)

Class(<Section > partial ...)

Class(<Course > partial ...)

Class(<Name > partial ...)

Class(<Address > partial ...)

Class(<Transcript > partial ...)

Class(<Post- Graduate> partial < Academic-Staff > ...)

Class(<Post- Graduate> partial <Graduate-STUDENT > ...)

Class(<Academic-Staff> partial <Staff> ...)

Class(<Graduate-STUDENT> partial <STUDENT> ...)
```

Figure 9-1: Partial of University Domain Expert Ontology

9.3.3 Experimental Evaluation

In this section the evaluation concentrates on different aspects of building ontology structure, which can be obtained from SQL, including creation of classes, subclasses, object and data type properties, as well as, properties domain and range; in addition to determining the cardinality restriction. However; some axioms cannot be obtained directly from SQL syntax, i.e. all Values From, Some Values From, Intersection of, which need to be verified by a domain expert. This explains the reasoning behind avoiding addressing them with all approaches; as we were able to reduce the range of our discussion to OWL commands that can only be obtained by SQL or mentioned by any one of participant approaches.

The domain expert, while converting the University database, decided to generate two ontologies. The first one is restricted to database specification (Specific Domain Ontology –SDO-); and the second one generalises some classes and enhances the overall ontology (Enhanced Domain Ontology –EDO-). An example of enhancing class is the Person class which is a generalised class representing the common elements between Student and Staff classes. Another method of enhancing involves creating a class for composite attributes. For instance the attributes Name and Address will have their own classes. Indeed generalisation carried out by the domain expert is valid, since this does not modify the originality of the database, although it cannot be obtained from SQL-DDL; however, we have incorporated both SDO and EDO ontologies in the evaluation process with all participant approaches.

9.3.3.1 Class Creation Evaluation

Figure 9.2 shows the class hierarchy of the enhanced and restricted domain expert ontologies, in addition to the participant approaches and class hierarchy results.

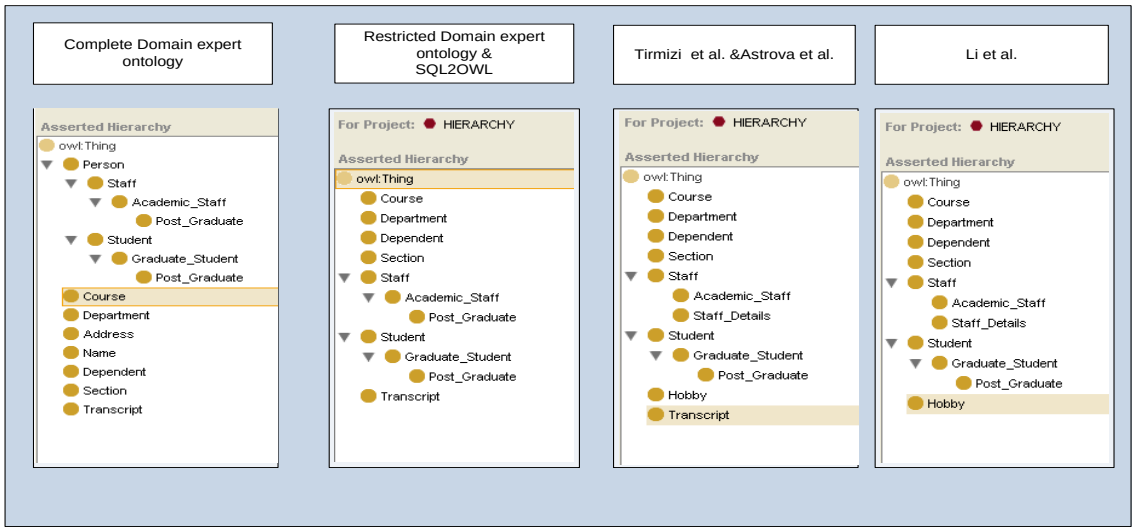


Figure 9-2: The Different Hierarchy Results for the University Ontology Experiment

From Figure 9.2 we can conclude that no approach is able to predicate the generalised class Person, since database attributes' names are often abbreviated and that causes difficulty in deducing the meaning of those names. For instance the attribute St-Id from the Student table differs from the attribute St-Id meaning Staff table and each one of these contains a different and significant meaning; another example of misleading attributes' names is that of the attribute SSN and the attribute National-Id which have the same meaning with different names. Therefore there is no system for acquiring generalisation between tables beyond user interference. In addition, the database designer does not consider composite attribute cases when structuring the relational model. They pick only its leaf components as attributes. Therefore there is no way to capture composite attributes from SQL-DDL. Table 9.5 concludes the correction and error results from applying class creation rules for each approach compared to restricted domain expert ontology SDO.

Table 9-5: Result of Class Creation Comparison

Approach criteria		Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
Class	Fragmentation	More class (staff-details)	More class (staff-details)	More class (staff-details)	☑
	Hierarchy	More sub- class between staff, staff-details	More sub- class between staff , staff-details	More sub- class between staff, staff-details	☑

Approach criteria		Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
	Multiple inheritance	☑	☑	☑	☑
	Multi-valued	One more class (Hobby)	One more class (Hobby)	One more class (Hobby)	☑
	Binary (M-M) relationship with additional attribute	☑	One class missing (Transcript)	☑	☑
	Ternary and higher relationship	☑	☑	☑	☑

From the above table we can observe that all three approaches fail to override the fragmentation problem. We can also see that all of them treat a multivalued table as a class. According to Li *et al.*'s approach, the fragmentation rule and hierarchy rule share the same conditions, which in this situation requires user involvement to decide which rule is applicable. If we choose a fragmentation rule as a default, the approach will result in failure. This is because all three tables; Staff, Academic –Staff, and Staff-Details will be assembled into one class, and Student, Graduate-Student, and Post-Graduate tables will be represented by one class also. Thus, we decided to omit use of the fragmentation rule and apply only the hierarchy rule in order to prevent user intervention. A useful general rule, to which any approach should adhere to succeed, is "no two rules are applicable for a relation at the same time".

9.3.3.2 Object property creation evaluation

Table 9.6 provides explanatory comparison between object properties creation and the criteria for evaluation approaches. Some approaches represent the foreign key with two inverse object properties and others only employ one. Our evaluation is concerned only with whether the foreign key is represented by an object property or not. The numbers do not matter in this case. The only exception is for the Many-To-Many relationship case, which should be represented by only two inverse object properties.

From the table we can observe that Astrova, *et al*.'s approach fails as a result of its generalisation in terms of treating each unary relationship as either symmetric or transitive; and as proven in Chapter 6 methods for obtaining such information cannot be learned from SQL directly. This requires domain expert involvement. Additionally, this approach did not succeed in defining the object property to represent the foreign key part of *n*-ary relationship's primary key. Put another way; forcing the object property to be *InverseFunctional* will ensure the domain and range will have a One-To-One relationship. However the foreign key here only represents part of the primary key. Therefore, the uniqueness exists only for whole primary key composite attributes which cannot be obtained by OWL. So we can only restrict the primary key to be Not Null by assigning it a cardinality of one. The approach of Li, *et al*. fails as it states that all foreign keys will have minimum cardinality of one. As is obvious, some foreign keys could have null value. Tirmizi, *et al*. describe an approach which contains contradictory actions, specifying an object property to be *Functional* and at the same time imposing it with cardinality of one. As is known, *Functional* means a minimum cardinality of zero and a maximum cardinality of one.

Table 9-6: Result of Object Property Creation Comparison

Approach **criteria**		**Astrova, et al.**		**LI ,et al.**		**Tirmizi, et al.**		**SQL2OWL**	
Object Properties	Binary (M-M)relationship	2 inverse OWL Object Property	☑	2 inverse OWL Object Property	☑	2 inverse OWL Object Property	☑	2 inverse OWL Object Property	☑
	Ternary and higher relationship	OWL Object Property InverseFunctionalProperty	☑	OWL Object Property minCrd=1, maxCrd=1,	☑	OWL Object Property Functional Property	☑ ☒	OWL Object Property, Card =1	☑
	Unary relationship	OWL Symmetric Property OWL Transitive Property	☒	OWL Object Property	☑	OWL Object Property OWL Cardinality of 1	☑ ☒	OWL Object Property Functional Property	☑

Approach **criteria**		**Astrova, et al.**		**LI ,et al.**		**Tirmizi, et al.**		**SQL2OWL**	
	Default foreign key	OWL Object Property	☑	OWL Object Property minCrd=1	☑ ☒	OWL Object Property	☑	OWL Object Property	☑
Object Properties characteristic	Not null	minCrd=1,	☑	minCrd=1,	☑	Card =1,	☑	minCrd=1,	☑
	unique	InverseFunctionalProperty	☑	maxCrd=1,	☑	Functional	☒	maxCrd=1,	☑
	Not null and unique	minCrd=1, InverseFunctionalProperty	☑	maxCrd=1, minCrd=1,	☒	OWL Object Property OP Functional, Card =1 Invers OP(Functional)	☒	subclass	
	weak entity	OWL Object Property Card =1,	☑	OWL Object Property Card =1,	☑	OWL Object Property	☑	OWL Object Property Card =1,	☑
	FK ⊂PK (N-ary)	OWL Object Property InverseFunctionalProperty minCard =1,	☑ ☒ ☑	OWL Object Property minCrd=1, maxcrd=1,	☑	OWL Object Property OP1 Functional, Card =1 OP2Functional	☑ ☒	OWL Object Property Card =1,	☑

All the corresponding object properties for all foreign keys in the University database example are summarised in Table 9.7. Indeed; the foreign keys participating in hierarchy or fragmentation or multivalued tables are not present in this table since they are handled by another class rule.

Table 9-7: Object Property of University Ontology

Object	**Domain**	**Range**	**Database restriction**	**Functional**	**Inverse**	**Card =1**	**Symmetric**	**Transitive**
Has Name	person	Name	Not null			☑		

Object	Domain	Range	Database restriction	Functional	Inverse	Card =1	Symmetric	Transitive
Has Address	person	Address	Not null			☑		
Co-offer	course	course					☑	☑
manger	staff	staff		☑				
Has student transcript	transcript	student	FK ⊆PK			☑		
Has course transcript	transcript	course	FK ⊆PK			☑		
Has student section	section	student	FK ⊆PK			☑		
Has ac-Staff section	section	ac-Staff	FK ⊆PK			☑		
Has course section	section	course	FK ⊆PK			☑		
Has staff	department	staff		☑				
Has course	department	course		☑				
Student belong dept	Student	department		☑				
Student register	Student	course	FK ⊆PK		☑			
Course register	Course	Student	FK ⊆PK		☑			
Has dependent	staff	dependent		☑				

9.3.3.3 Datatype property creation evaluation

The attribute in the database can appear in one of the following shapes, single valued or multivalued, simple or composite and the attribute can be accompanied by various combinations of null and uniqueness restrictions. Table 9.8 illustrates the reaction towards each attribute's criteria using a correct or false evaluation for participant approaches.

Table 9-8: Result of Datatype Property Creation Comparison

Approach **criteria**		**Astrova, et al.**		**LI ,et al.**		**Tirmizi, et al.**		**SQL2OWL**	
Datatype Properties characteristic	Not null	minCrd=1,	☑	minCrd=1,	☑	Card =1,	☑	minCrd=1,	☑
	Unique	InverseFunctionalProperty	☒	maxCrd=1,	☑	Functional	☒	maxCrd=1,	☑
	Not null and unique	minCrd=1, InverseFunctionalProperty	☑ ☒	maxCrd=1, minCrd=1,	☑	Functional	☒	Card =1	☑
	Primary key	minCrd=1, InverseFunctionalProperty	☑ ☒	maxCrd=1, minCrd=1,	☑	Functional	☒	Card =1,	☑
	Check in	oneOf	☑	==	☒	oneOf	☑	oneOf	☑
	Check	Has Value	☒	==		==		==	
	Primary key in two tables	subclass	☑	unionOf	☒	Subclass	☑	Subclass	☑
	Multivalued	Functional	☒	Functional	☒	Functional	☒	Non functional	☑
	Default attribute	Maxcar=1	☑	==	☒	Functional	☑	Functional	☑

Interestingly, Astrova *et al.*'s approach is different from the other approaches in defining some attribute characteristic. For instance it uses the functor (*Inverse Functional Property*) to restrict datatype to UNIQUE or NOT NULL and UNIQUE; the reason behind this is an approach targeting OWL-Full. Therefore their result cannot be checked for inconsistency by OWL reasoner. Another problem arises in Astrova *et al.*'s approach in determining the result of the all check constraint. However we will prove that there is no general rule for handling different methods for using check constraints.

Proof:

Here the term check in SQL can occur in different cases.

- **Case 1: Check has a value**

Check can have a specific value and it could be represented by a (*hasValue*) term in the ontology structure. The example below shows an attribute and it is analogous OWL segment.

```
Create TABLE Customer

type Varchar CHECK (type='Software'))
```

And the corresponding OWL will be:

```
DatatypeProperty(<type > Functional domain(<Customer>) range(hasValue "Software") range xsd: string))
```

- **Case 2: Part of XML domain**

The check constraint can limit the domain of the attribute with a condition. This is the SQL sentences

```
Create TABLE Customer
(SID integer CHECK(SID > 0),
```

And the corresponding OWL will be:

```
DatatypeProperty(<SID> Functional domain(<Customer>) range(xsd: positiveInteger))
```

- **Case 3: Nested condition**

In this case, the example below shows the usage of check in more than one condition.

```
Create TABLE Person
(P_Id int Not Null,
City varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Leicester'))
```

Here there is no equivalent OWL segment. Consequently, we proved that there is no way to limit the use of check with a specific rule. Table 9.9 demonstrates all data type

attributes and their characteristics in both database and ontology sides for the University example.

Table 9-9: Datatype Property of University ontology

Datatype	Domain	Range	Datatype Characteristic	Pk	Not null	Unique	Not Null& Unique
Dept_id	department	int	Card=1	☑			
Dept Name	department	string	minCard=1		☑		
Dept Phone	department	string	Functional				
n-id	person	int	Card=1				☑
Family name	person	string	minCard=1		☑		
Mid-name	person	string	Functional				
First Name	person	string	minCard=1		☑		
Staff-id	staff	int	Card=1	☑			
DOB	staff	date	Functional				
Email	staff	string	Functional				
Ext phone	staff	string	Functional				
Home phone	staff	string	Functional				
Post held	Academic- staff	string	One of				
Specialty	Academic- staff	string	Functional				
Student id	Student	int	Card=1	☑			
Sex	Student	string	Functional				
Module name	Student	string	maxCard=1			☑	
Hobby	Student	string	===				
Research area	Graduate-Student	string	Functional				
Project group	Post- Graduate	string	Functional				
Dependent id	Dependent	int	Card=1	☑			

Datatype	Domain	Range	Datatype Characteristic	Pk	Not null	Unique	Not Null& Unique
Dependent name	Dependent	string	minCard=1		☑		
Course id	Course	int	Card=1	☑			
Course name	Course	string	minCard=1		☑		
Course credit hour	Course	int	Functional				
Grade	transcript	int	Functional				

9.3.4 Score scheme evaluation

9.3.4.1 The Quality of Score scheme

The scoring scheme is a criterion that can be used to evaluate the correctness of the ontology design. In the Scoring Scheme, each ontology context (classes, relationships and datatype properties) will have a specific score.

We focus on the class and the object properties context in ontology structure with considering the overall quality of the ontology. Each subject approach generates an ontology with same specification source which can be measured in term of its classes and relationships. The domain expert is responsible for developing the correct solution for each case in order to assess the subjects' ontologies against a benchmark. The Scoring Scheme is a simplified procedure that utilises the score mechanism and assesses the ontology model quality. Based on the problem space description, the Scoring Scheme therefore identifies the correct and suitable match for ontology constructs, including classes and relationships. Although the quality of ontology model relies on many different factors of modelling design, many researchers address that by adding each single context score to the total score in order to reflect that the overall quality is incorrect, since the total score does not consider the absence the construct validity.

However, we overcome this issue in our evaluation by applying the following criteria:

- The analysis of model quality is managed for both individual context, and the connections between ontology contexts. The structure model of the ontology is

built on these connections. Therefore connections between classes such as subclass, and object properties are highly scored in our evaluation system. In addition, our evaluation considers connections between datatype attributes and their classes by scoring their domain. Therefore our evaluation handled both individual context and design context.

- Before the matching process take place, each ontology output is validated using OWL reasoner to remove any inconsistence structure.
- A single measure is not sufficient to measure the quality of a model; therefore there is a need for a composite measure for different ontology contexts.
- We maintain the evaluation, using the OWL syntax model, which can obtain an understanding of the ontology, to assist in choosing appropriate classes and relationships. Moreover, applying OWL syntax during the evaluation process helps us to identify correct or relevant class-relationship constructs easily.

9.3.4.2 Matching Process:

After generating target ontologies from each approach, the matching process starts between the approaches' ontologies and the domain expert ontologies. Each ontology model was compared to two domain ontologies. Therefore each approach output goes through one matching with SDO and another matching with EDO to generate their matching scores. During the matching process each approach output was examined to determine the correct match with the domain ontology, and simultaneously the missing or superfluous context will affect the total score. In other words for each context, We endeavour to achieve similar pair matching, and the context which does not have a corresponding match in the benchmark can be considered as superfluous.

Each approach may have alternative solutions; therefore they have been tested, and the nearest to best solutions have been selected in our experiments. This makes the evaluation equal across all the approaches.

9.3.4.3 Score Method

Each output ontology was measured according to the steps below:

- For every correct class or relationship (object property) included in the ontology, subject to the evaluation, one point is awarded.
- For every correct datatype property included in the ontology, 0.5 point is awarded.
- For every correct representation for a property (object or datatype) included in the ontology, 0.25 point is awarded.
- If the approach fails to identify a class or a relationship including a subclass relationship that should be part of the ontology, one point is subtracted from the score.
- If the ontology structure contains superfluous classes or superfluous relationships, a penalty of −0.5 points is assessed, whereas for a superfluous datatype property the penalty will be −0.25 points.
- If the Approach identifies incorrect representation for a property, −0.25 point is applied.

See Tables 9.10 and 9.11 for awarded and subtracted points respectively.

Table 9-10: Score Points Awarded

Facet	Correct Points	Maximum Possible Points
Class	**+1**	**+1**
Subclass	**+1**	**+1**
Relationship (object property)	**+1**	**+1**
Datatype property	**+0.5**	**+0.5**
Property representation	**+0.25**	**+1.25**

Table 9-11: Score Points Subtracted

Facet	Error Classification and Points			Maximum Possible Points
	Major Error (−1.0 Points)	Medium Error (−0.5 Points)	Minor Error (−0.25 Points)	
Class	Class Missing	Superfluous Class		-1.5
Subclass	Subclass Missing	Superfluous Subclass		-1.5
Relationship (object property)	Relationship Missing	Superfluous Relationship	Incorrect representation	-1.75
Datatype		Datatype Missing	Incorrect representation	-0.75

In the table we can observe that property is given to include both object and datatype. The common representation of property includes domain, range, different type of cardinality, and functional. However there are some representations that are applied only to object property such as symmetric, transitive, inverse, and inverse functional whereas for datatype property representation include the term “one of”. The common elements between object and datatype properties are domain, and range. All these criteria are countable for the purposes of our evaluation.

9.3.4.4 Data Analysis and Results from the Experiment

The analysis was obtained manually for each approach and two score outputs given for each approach. The first result score is shown in Table 9.12 after the comparison occurs between the subjects and the strict domain ontology.

Table 9-12: Scoring Scheme compared to SDO

Approach / Facet	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL	Specific domain ontology
class	10.5	8.5	10.5	14	14
Object property	11	11.5	11	14	14.5
Datatype property	16.75	12.5	15	19.5	19.5
Total	38.25	32.5	36.5	47.5	48

Figure 9.3 depicts the comparison of three criteria of the ontology including classes, object properties and datatype properties and their characteristics. The figure shows that SQL2OWL achieves the highest scores in all three fields of comparison. Thus proving that our transformation system is better.

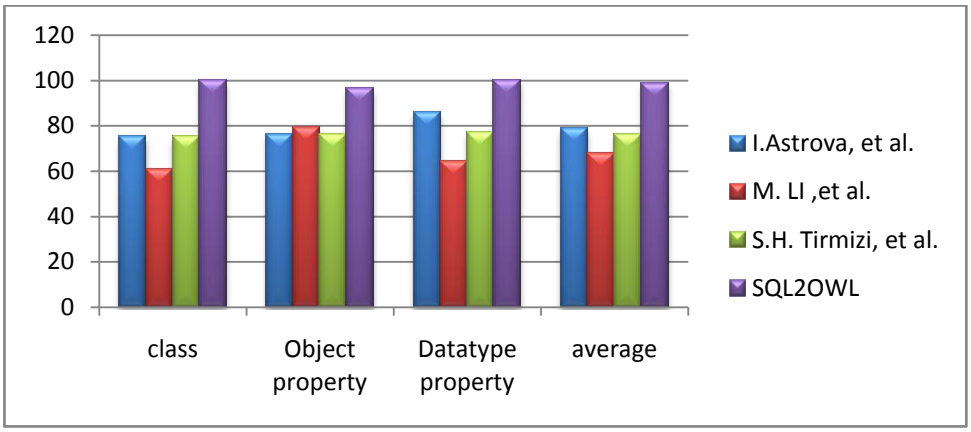


Figure 9-3: subject approaches result compared to SDO

Figure 9.4 shows the comparison between the subjects and the enhanced domain ontology, which contains extra classes and relationships. Since capturing these new generalised concepts is impossible, without background knowledge, SQL2OWL cannot discover the generalised class (Person) between Staff and Student classes. Although SQL2OWL proved that it could obtain the best score when compared to the other approaches, even with enhanced domain ontology.

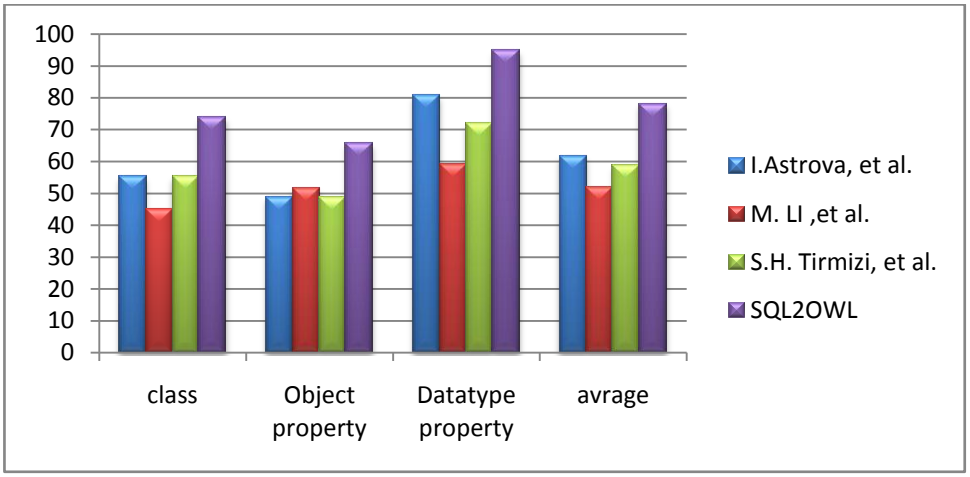


Figure 9-4: subject approaches result compared to EDO

The exact score for each subject is shown in Table 9.13 and the full details of how these scores were obtained for all contexts in University ontology is shown in tables in Appendix D. This score scheme substantiates that SQL2OWL has a decent performance in determining the structure ontology model, especially classes and relationships.

Table 9-13: Scoring Scheme compared to EDO

Approach / Facet	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL	Enhanced domain ontology
class	10.5	8.5	10.5	14	19
Object property	8.5	9	8.5	11.5	17.5
Datatype property	15.75	11.5	14	18.5	19.5
Total	34.75	29	33	44	56

9.3.5 Precision, Recall, and F-measure

The three techniques are part of a compliance measure which is considered as standard information retrieval metrics which can be used in evaluating the results of different approaches in order to specify the degree of matching. *Precision, recall* and *F-measure* are widely accepted and considered. Their role is to provide estimation values for the quality matching process. It is necessary here to provide brief details about how these measures were calculated.

9.3.5.1 Precision:

Precision is a function to measure the amount of correct matching discovered, versus the total amount of mapping achieved by a particular approach. Given a reference context R, the precision of some matching A is a function $P : \Lambda \times \Lambda \rightarrow [0, 1]$ such that:

$$Precision = \frac{\text{number of correct matchings found}}{\text{number of matchings retrieved by a certain approach}} \times 100\%$$

Alternatively, it can be defined as follows:

$$Precision = \frac{|A \cap R|}{|A|} \times 100\%$$

where R is the set of context with correct reference matching and A is the set of all matching retrieved by a particular approach. Gaining the value of 1 in the precision function only indicates that all obtained contexts find a correct match; however this does not imply that all the target contexts have a corresponding match. Therefore, in order to balance the precision function, we suggest using the recall measure. Here Table 9.14 and Table 9.15 showed the precision in 100 percent of each subject approach compared to SDO and EDO respectively.

Table 9-14: Precision compared to SDO

Approach	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
Class &subclass relationship	13/16	12/15	13/16	15/15
Object	13/14	13/14	13/14	13/13
Object domain	12/14	8/14	12/14	13/13
Object characteristic	10/18	12/13	7/11	13/13
Datatype	26/26	25/25	26/26	26/26
Datatype domain	25/26	24/26	25/26	26/26
Datatype characteristic	25/31	12/13	18/25	25/25
Precision SDO total	124/145	106/120	114/132	131/131
Precision SDO	85%	88%	86%	100%

Table 9-15: Precision compared to EDO

Approach	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
Class	13/16	12/16	13/16	14/14
Object	12/15	12/15	12/15	12/14
Object domain	12/15	8/15	12/15	12/14
Object characteristic	9/22	9/14	7/20	15/15
Datatype	26/26	25/25	26/26	26/26
Datatype domain	21/26	20/26	21/26	22/26
Datatype characteristic	12/25	24/25	18/25	25/25
Precision EDO	105/145	110/136	109/143	126/134
Average	72%	80%	76%	94%

9.3.5.2 Recall

Recall is a function to measure the number of correct discovered matchings, versus the total correct context matchings and not limited to the matching obtained by the approach. Given a reference context R, the recall of some matching *A* is a function *R*: $\Lambda \times \Lambda \rightarrow [0, 1]$ such that:

$$Recall = \frac{\text{number of correct matchings found}}{\text{number of existing context}} \times 100\%$$

Alternatively, it can be defined as follows:

$$Recall = \frac{|R \cap A|}{|R|} \times 100\%$$

The drawback of the recall function is that it does not provide any indication of a false match number. So the high value reveals only the actual correct match found. Here Table 9.16 and Table 9.17 show the recall value for each approach compared to SDO and EDO respectively.

In most cases, the approach applied found all the contextual possibilities in the specific domain ontology. However it omitted the contextual mapping of symmetric and transitive for the Co-offer relationship and Astrova, *et al.*'s approach to capture it. At the same time Astrova, *et al.*'s approach failed when applying their general symmetric and transitive rules for defining the self relationship Manger. Due to this most approaches avoid defining a general rule for self relationship characteristics. From Table 9.16 we see both LI *et al.* and SQL2OWL have the best two recall numbers amongst the other approaches. Table 9.17 shows SQL2OWL is dominant in all different facets.

Table 9-16: Recall compared to SDO

Approach	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
Class &subclass relationship	13/15	12/15	13/15	15/15
Object	13/13	13/13	13/13	13/13
Object domain	13/13	8/13	13/13	13/13
Object characteristic	10/16	12/16	7/16	14/16
Datatype	26/26	25/26	26/26	26/26
Datatype domain	25/26	24/26	25/26	26/26
Datatype characteristic	25/25	12/25	18/25	25/25
Recall SDO total	125/134	106/134	115/134	132/134
Recall SDO	93%	97%	85%	98%

Table 9-17: Recall compared to EDO

Approach	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
Class	13/19	12/19	13/19	14/19
Object	12/14	12/14	12/14	12/14
Object domain	12/14	8/14	12/14	12/14
Object characteristic	9/15	9/15	7/15	15/15
Datatype	26/26	25/26	26/26	26/26
Datatype domain	21/26	20/26	21/26	22/26
Datatype characteristic	12/25	24/25	18/25	25/25
Recall RDO	105/139	110/139	109/139	126/139
Average	75%	79%	78%	90%

9.3.5.3 F-measure

The concept of the F-measure function is to provide an efficient single measure that represents the combination of *precision* and *recall*. It is also utilised in measuring approach performance:

$$\mathrm{F-measure} = \frac{2 \times \mathrm{precision} \times \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \times 100\%$$

The result of the F-measure and the other measures were demonstrated in Table 9.18 for all four approaches, and the average result was compared to specific domain ontology for all three measures.

Table 9-18: F-measure compared to SDO

Approach	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
Precision SDO	85	88	86	100
Recall SDO	93	97	85	98
F-measure SDO	88	92	85	99

The outcomes shown in Table 9.19 also reveal that there are considerable differences between the measures for each approach, compared with SQL2OWL.

Table 9-19: F-measure compared to EDO

Approach	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
Precision EDO	72	80	76	94
recall EDO	75	79	78	90
F-measure EDO	73	80	77	92

Figure 9.5 depicts all three measures including Precision, Recall, and F-measure for subject approaches where the benchmark is SDO; whereas Figure 9.6 used EDO as the benchmark. The evaluation example was created to test each subject to see if it had overall model quality. It also shows that each context has a significant influence on its model.

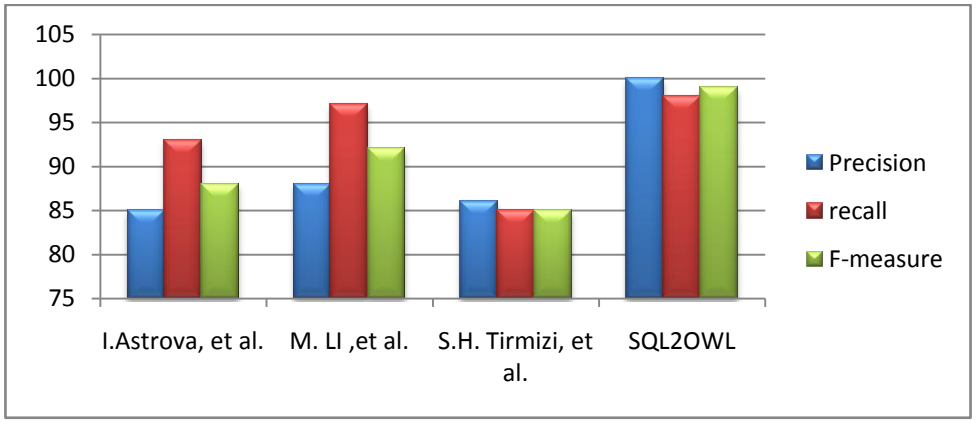


Figure 9-5: result of the approaches compared to SDO

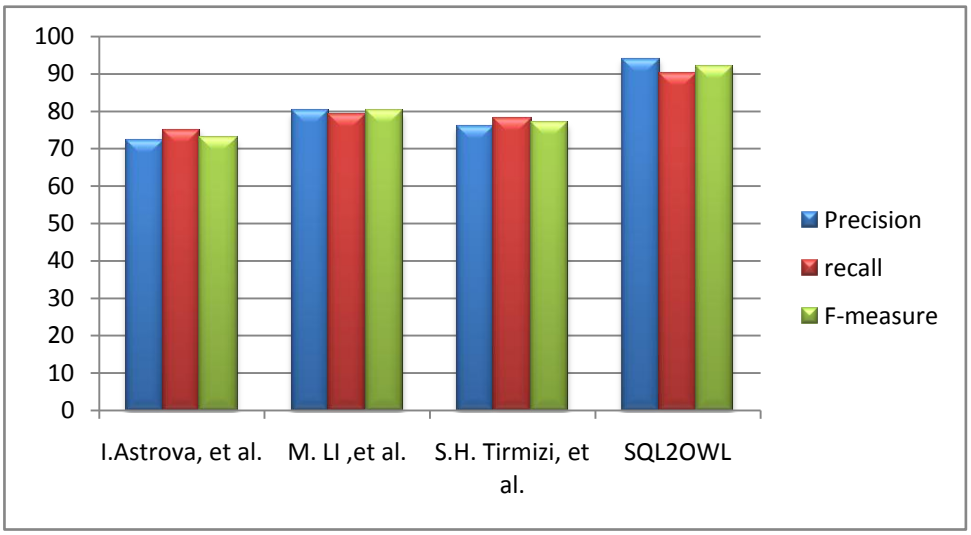


Figure 9-6: result of the approaches compared to EDO

9.4 Quality of Transformation

The last section illustrated that our transformation system is total and comprehensive; since its rules include all the possible cases of relations. This final test will exploit another technique to test the quality of the system rules. To do this a reverse transformation from ontology to database is done. If the reverse transformation can be guaranteed to reproduce the original database source then the system is correct. Therefore the inverse database must include all original components extracted by analysing tables, attributes, relationships, and constraints. Here we assume only the information maintained by the transformation process can be reproduced. A trigger is an example of non transformed information.

Indeed any transformation process includes some loss of semantic. It is expected that as a result of this rounded transformation not all information will be preserved. For instance the fragmentation tables convert to one class and after reverse transformation all the information will be appear in one table. We do not consider this to be an error as this is the right representation for a single entity. Another problem relates to specifying the primary key of a table. If the table has an alternative key, which can be characterised by UNIQUE and NOT NULL constraints, then the system would not distinguish

between a primary key and an alternative key. This is because both datatype properties will have a cardinality of one constraint.

There is an algorithm suggested by [63] for transforming from ontology to database. However we cannot utilise it since the goal of our reverse transformation is different from the target of this algorithm.

9.4.1 Reverse Transformation Algorithm:

Generally:

- All classes convert to tables.
  This step includes all classes corresponding to strong entity, weak entity, *n*-ray relationship and Many-To-Many with additional attributes relationship.
- Both the two inverse object properties in two different classes will form a table. And this table will represent the Many-To-Many relationship case.
- If a datatype without cardinality constraints that is not *Functional* creates a table for a multivalued attribute and makes it part of the primary key, this will also create a link between a multivalued table and the original table by making the primary key of the original table part of this new table's primary key. And at the same time make it would become a foreign key.
- All datatype with a minimum cardinality of one is assigned the corresponding attribute NOT NULL.
- All datatype with a maximum cardinality of one is assigned the corresponding attribute UNIQUE.
- All datatype with a cardinality of one assigns the corresponding attribute primary key. If there is more than one in the original class pick one as a primary key and the assign the others as NOT NULL and UNIQUE.
- All object properties with cardinality = 1 generate a foreign key that is part of the primary key for the corresponding table.
- If object property has a minimum cardinality of one assign the corresponding foreign key NOT NULL.
- If the object property has maximum cardinality of one assign the corresponding foreign key UNIQUE.

- For each subclass generate a foreign key to act as a primary key in the child table, referencing the primary key of the parent table.

9.4.2 The Lexical Overlap Measure

The idea of this test was mentioned by [39] theoretically; however, the approach does not produce any outcome for this measure to compare it with our results, since it does not apply it. As has been stated, not all relational database model constructs can be transformed to an ontological model. Therefore we need to see how much semantic loss occurs during the transformation process.

In a formal way,

- Let T_1 be the transformation of a relational database R_1 to ontology O.
- Let T_2 be the reverse transformation of the ontology O to an inverse relational database R_2.

That is, $T_1(R_1) \rightarrow O \rightarrow T_2(O) \rightarrow R_2$

The transformation T_1 could be considered as reversible if it assures the equivalence between the two relational databases R_1, R_2. That is, $R_2 \equiv R_1$.

If a lexical overlap measure [54] is denoted as $L(R_1, R_2)$, it produces a value of 1, then we can say R_1 and R_2 are *equivalent.*

That is, $L(R_1,R_2) = 1 \Rightarrow R_2 \equiv R_1$.

The lexical overlap measure is calculated as follows: $L(R_1, R_2) = |L_1 \cap L_2| / |L_1|$

Where

- L_1 is a set of all constructs in the relational database R_1 and,
- L_2 is a set of all constructs in the relational database R_2.

Table 9.20 shows original database constructs and their matching ontology and the inverse database.

From the table:

- $L_1 = 56$, $L_2 = 52$
- $L(R_1, R_2) = 52/56 = 0.92$

The missing Table includes Staff-details and its primary key foreign key is St-Id. The change of the domain for the attributes of staff-details table is not counted, since we consider the fragmentation tables to originally represent one entity; in addition no attribute is missing. Attributes N-Id or Staff-Id can represent the primary key of the Staff table after reverse transformation. Here the system picks one of them randomly.

Table 9-20 : Original Database and Inverse Database Constructs

Database context	No	Ontology context	No	Inverse Database	No
Table for strong entity + Fragmentation table	6+1	Class	6	Table	6
Table Weak Entity	1	Class	1	Table	1
Table for M:M relationship with two FK	1	Object property	2	Table +2 FK	1
Table for M:M relationship with attribute	1	Class	1	Table+2 FK	1
Table for N-ary relationship	1	Class	1	Table +3 FK	1
Foreign key (subclassof)	4	Sub class of	4	Sub class of	4
Foreign key attributes (not part primary key)	6	Object property	6	Foreign key	
Primary key attribute (not foreign key)	6	Datatype with cardinality of one	6	Primary key	6
Multi-valued attribute	1	Datatype	nil	Multi-valued Table	1
Non key not null	5	Datatype with mincard =1	5	Non key not null	5
Non key unique	1	Datatype with maxcard=1	1	Non key unique	1
Check in	1	One of	1	Check in	1
Non key	12	Functional	12	Non key	12
Non key not null and unique	1	Datatype with cardinality of one	1	Non key not null and unique	1
Foreign key attributes (not part primary key)	6	Object property	6	Foreign key	6

9.5 Completeness of Transformation

The idea of completeness for our transformation rules is based on specifying all the possible range of relations described by the relational model. In addition to that our rules have to cover different types of relationship that exist between these relations. Our system can successfully identify all types of relations and relationships; therefore the transformation rule can then partition those relations such that each relation accommodates precisely one rule which implies that no relation satisfies more than one rule, nor is a relation not applying any rule from the class creation rules. This would ensure the completeness of our system.

The basic idea of translating a relation to a class is trivial, without considering the challenges of the variety of relationship types for any transformation system. A relationship based on a cardinality ratio appears as One-To-One, One-To-Many or Many-To-Many. Since each relation might have more than one relationship; there are multiple foreign keys in SQL schema, which makes relations appear somehow complicated. The mutual action between the primary keys and foreign keys of relations provides clues to creating classes and relationships as well as enriching relationships with their characteristics.

Hypothesis:

The space of relations described in the relational model using key combinations include the interaction of primary key and references between relations represented by the foreign key which can be classified in 10 disjoint situations.

Proof:

The proof contains applying a closure operation to relation space and enumerated syntactic to the possible cases.

First, the space of different possibilities is partitioned according to the number of foreign keys contained in relations. This means that each relation will be examined and

then categorised into the space of relations in agreement with its foreign key number. Figure 9.7 will provide a notion of various possibilities for relation space.

Preface:

The primary key can have one or more attributes (composite key). Also the foreign key can have one or more attributes. The database relation has only one primary key and any number of foreign keys.

Predicates:

- PK: is a single attribute primary key of a relation.
- C-PK: is a composite primary key of a relation C-PK can contain x amount of attributes where $x \geq 2$.
- S-FK: is a single foreign key for a relation.
- N-FK: a relation with at least two foreign keys.

Notice that there is a distinction between single foreign key and foreign key with single attribute. We consider the number of foreign keys in a relation, and the number of attributes does not matter. However this is not applicable to the primary key since each table has only one primary key.

Each relation only has one primary key. This primary key does not necessarily represent only one attribute since the primary key can be a composite attribute.

Initially we can start with:

E′ → E

E → PK+T | C-PK +T

E → S-FK

E → N-FK

T → S-FK | N-FK

The entity schema contains many entities, and each entity can be represented by a table with one primary key (single or composite). In addition the entity can have one single foreign key or N number of foreign keys.

Then by applying the closure operation of the LR(0) item set, the following elements are obtained:

1. PK + S-FK: a relation has a Primary Key and only one Foreign Key

    a) PK $\cap$ S-FK = 0: the Foreign Key and the Primary Key do not share any attributes.

    b) PK = S-FK: the Foreign Key is the Primary Key

2. PK + N-FK: a relation has a Primary Key and at least two or more Foreign Keys

    a) PK $\cap$ N-FK = 0: the Foreign Key and the Primary Key do not share any attributes.

    b) PK $\subset$ N-FK: one of the Foreign Keys is also the Primary Key.

3. C-PK + S-FK: a relation has a Composite Primary Key and only one Foreign Key.

    a) C-PK $\cap$ S-FK = 0: the Foreign Key and the Primary Key do not share any attributes.

    b) S-FK $\subset$ C-PK: the Foreign Key is part of the Primary Key for weak entity and multivalued.

4. C-PK + N-FK: a relation has a Composite Primary Key and at least two or more Foreign Keys.

    a) C-PK $\cap$ N-FK = 0: all the Foreign Keys and the Primary Key do not share any attributes.

    b) N-FK $\subseteq$ C-PK: all the Foreign Keys are part of the Primary Key.

    c) C-PK $\cap$ N-FK $\neq$ 0, C-PK – N-FK $\neq$ 0, N-FK – C-PK $\neq$ 0: The Foreign Keys and Primary Key share common attributes.

Sequeda's report [53] also proved that the combination between a primary and foreign key can have ten cases above. However it fails when applying these cases because the problem solving process is based on a false generalisation. The approach for any relation assumes there are more than two foreign keys representing *n*-ary relationship. Also the approach does not cover the entire spread of relations since the case of the single foreign key part of a composite primary key covers only a weak entity table and does not cover the multivalued attribute relation.

Figure 9.7 shows all ten possible combinations of primary key and foreign keys. The eleventh case represents a relation without foreign keys. All rule sets must apply property characteristics, which can be obtained from SQL aspects. For attributes and foreign keys characteristics, we apply rule 16 and rule 17 respectively. The space of the relation in Figure 9.7 utilises the assumption of the existence of complete and correct specification for fragmentation. However if we want to use the record count technique then the pair rule (1, 2) will be replaced by the pair rule (6, 7) in order to deal with the fragmentation case. The pair rule (8, 5) substitutes the rule (3, 5) for the hierarchy rule case; the figure includes the OR circle which means one of the two cases is true based upon another relation specification. Therefore we have to apply more conditions in order to specify if the case is designed for hierarchy or fragmentation. Therefore the solution is based on the referenced relation interaction. In addition to the condition $FK_i = PK_i$ the differentiation rules will have the following conditions:

- Fragmentation condition: $FK_j=PK_j$ and FK_j refer to PK_i and FK_i refer to PK_j.
- Hierarchy condition: we have two cases
    - If $FK_j \neq PK_j$ and FK_j refer to PK_i then the relation of PK_i is master and the other one is subclass from it.
    - If $FK_j=PK_j$, but FK_j do not refer PK_i then this case of a multiple inheritance.

It is also evident that mainly rule sets 11, 12 are used, since they represent the class and datatype default rules. The cases, such as tables from strong entity, weak entity, and *n*-ary relationship are covered by these two rules. The rules also apply both hierarchy and fragmentation for any combination of foreign keys.

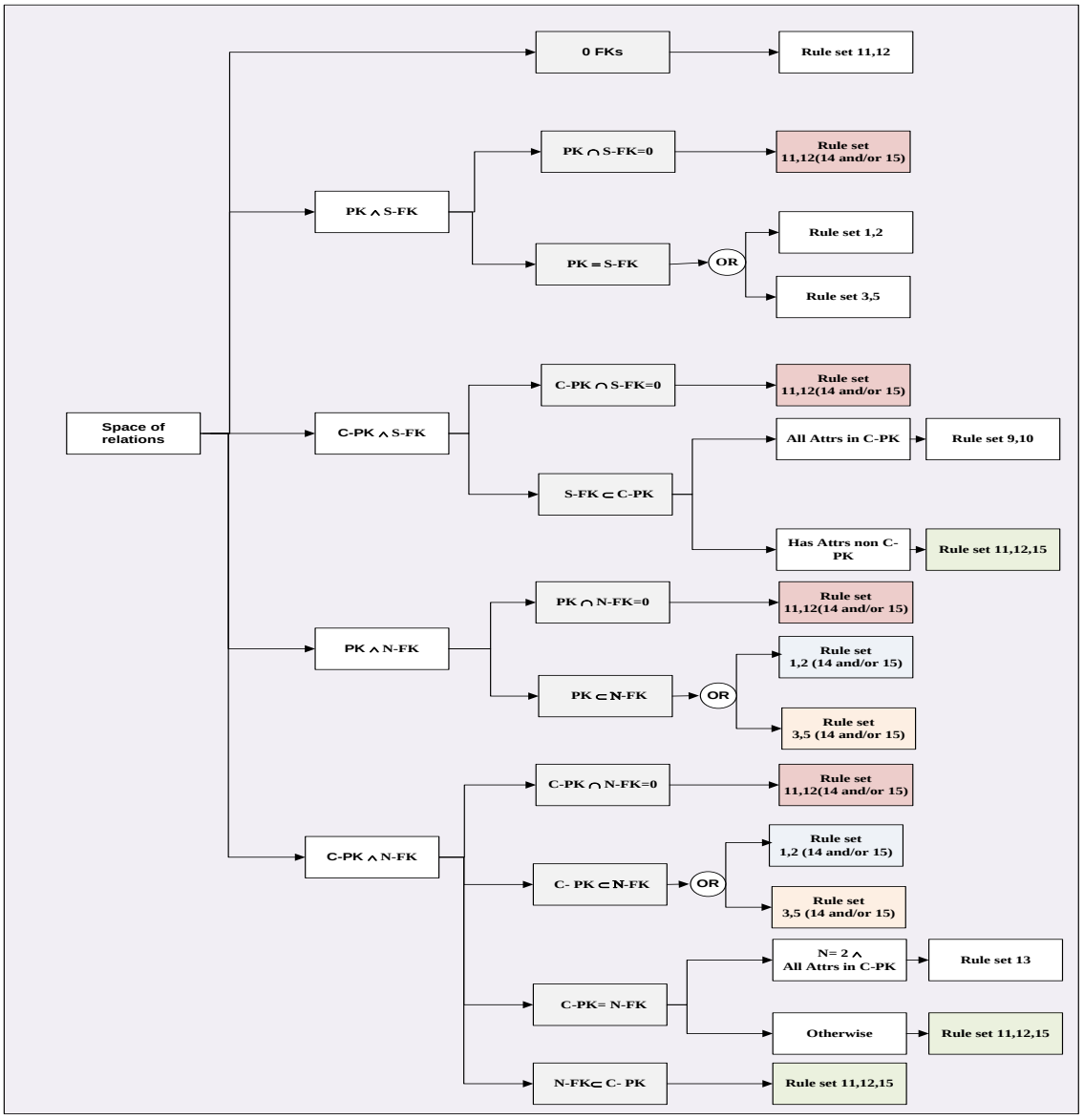


Figure 9-7 : The Space of Relation Tree

9.6 Other Measure of Success

For the success of our system we consider the following criteria.

- Formality;
- Accuracy and Correctness;
- Flexibility and Functionality;
- Scalability and efficiency.

The following section briefly describes how our system realises these criteria.

9.6.1 Formality

The use of formality helps us to clarify our system from ambiguous transformation rules. We use the well known notation of IF condition THEN action. Since this notation is easy to comprehend it makes implementation easier. Any transformation rules that incorporate lack of formality might confuse the reader, and create an uncertain system.

The only two approaches used to complete a formal notation are Stojanovic *et al.* [36], and Tirmizi *et al.* [45]. The former one is not capable of capturing richer semantics, since its target is RDFS; whereas the latter one used FOL expression which is easy to implement in Datalog interpreters or Prolog environment. However, it is not simple to comprehend the nested complex rules and therefore it is not simple to modify. As for other approaches they either use English descriptive rules [39, and 55] or they switch between some formal notation and the English language [41, and 43].

9.6.2 Accuracy and Correctness

To test the system's result against accuracy and correctness criteria we followed the steps below:

- A comparison between the correct representation of ontology and the human analysis to check the measure of accuracy.
- An inverse transformation of the outcome ontology to database schema to check the measure of correctness.
- Validate the result ontology using the reasoner to catch any inconstancy.

And all these steps accomplished by the techniques of Score scheme, Precision, Recall, F-measure, and the Lexical Overlap measure during the evaluation process.

9.6.3 Flexibility and Functionality

Another distinguished feature of our system is the method for adjusting the fragmentation rule applied based on the information given.

There are three options for applying the fragmentation rule as given below:

a. For a complete and correct specification database, where each table refers to the other the system we use a normal method.
b. If option one is not applicable then the system uses the count method if the meta-data for the database is available.
c. Treat fragmentation as a hierarchy for other cases.

Our system applies these three options in a sequence to search for tables referring to each other if not available; then uses a count method to calculate the record of each table participate in fragmentation rule. If the first option is not applicable and data is not offered then a One-To-One relationship is given, sharing the same primary key as the hierarchy; thus, proving our system's functionality, since it deals with a well designed database in the first instance. Whereas a badly-designed system would be managed according to the other two options. For flexibility criterion our system is domain independent, which means it will accept any type of relational database as input then build the corresponding ontology.

9.6.4 Efficiency and Scalability

One aspect that enhances the efficiency of our rules is database normalisation. This step is incorporated in the implementation process and can reduce the redundancy of the database sources before applying the transformation rules.

In addition, the order of our rules is significant. Following this order in applying the approach rules will enhance our tool performance and reduce the time required for generating the target ontology, by narrowing the set of tables applied for each rule, especially for large scale databases. Therefore the procedure for applying the transformation rules contains two steps in order:

a. Identifying tables sharing the same primary key. Then other tables that do not satisfy this condition will be eliminated from fragmentation and hierarchy rules. After that the system applies the fragmentation rule if it is applicable, otherwise the hierarchy rule is applied.

b. Identifying the two cases of tables that do not form classes then using the rest to convert to classes.
    i. Identifying tables of a Many-To-Many relationship without additional attributes which cannot be taken from a class.
    ii. Identifying tables with part of their primary key referring to other tables. Such weak entity tables and multivalued tables will be nominated and other tables will be eliminated. Then the tool applies a multivalued attribute condition.

The execution times vary from one database type to another. Here is the combination of database size and the rate of changes the time is based on:

- Small size database with frequent data change;
- Large scale database with frequently data change;
- Small size database with a few updates;
- Large scale database with a few updates.

The last two cases would not cause any problems. However, to solve the first two cases there are two different techniques for producing the final resultant ontology. The first technique involves transforming the database schema, while the data still resides in its original database. In this case the transformation time is counted in milliseconds. However querying the database through ontology might take time, since it translates a query from SPARQL to SQL Query language in a database; then the system will generate the answer instantly as needed. In this process the time factor is significant, although this process allows the content of databases to be accessed with acceptable response times as long as it is not exponential [60]. In another technique the system is generating an OWL ontology and populating the ontology with data instances in order to produce a knowledge base. In this process all the transformation is done off-line, here the time factor is not significant. For updates the system needs to reproduce its knowledge base periodically. Therefore time would not be considered as an obstacle. This insures the scalability of the system; therefore this technique has been applied to our system. Both techniques utilise a powerful database for storing large scale data. In fact there are two factors that might affect our system scalability:

- The number of databases participating in the approach simultaneously.
- The size of the database.

Since our approach processes only one database at a time, and the transformation is done off line, both factors would not affect our system. Therefore our system can be considered scalable.

There are many other factors that have been taken into consideration when testing the efficiency of our approach:

- Number of tables.
- Number of subsumption (IS-A).
- Number of mutual property (relationship).
- Data size (number of instance in all tables).

The number of tables in the database is not relevant for measuring the ability of the system's efficiency. However the number of relationships is the more effective factor, since all the rules have to check the relationship between tables. Indeed if the number of tables increases the number of relationships increases too. Additionally the type of the relationships is an important factor here. For example if the number of (one–to–one relationships is big then the time will increase, since we have proposed three different solutions regarding whether the database is well defined or not, although our system took an acceptable time to manage large scale database.

The system has been tested using different databases from different domains to confirm both its efficiency and the scalability. The sample contained databases from the ER conceptual model, which do not have class inheritance; meaning that relational design occurred and not hierarchal design. Another sample includes databases from an EER conceptual model.

9.7 Summary

The evaluation in this chapter goes through many steps. First of all we held a comparative study between our approach and other similar approaches. This comparison based on showing the ontology elements production in each approach. Also it showed

the uncovered cases from the database model in each approach or the fault solution based on wrong analysis for the specification of some database cases. Then the weighted criteria evaluation technique is utilised in order to show that not only the ontology elements are the most affect criteria in the transformation system. However there are other criteria such as availability which means that the semantics can be obtained easily from the source or not. Also the automation of the approach can affect system, since manual transformation system would be considered as building ontology from scratch. The final criterion that gets weighted is the formality of the approach. Another evaluation technique is score method that showed the superiority of our approach compared to other extant approaches. We utilise the compliance measures such as precision, recall and F-measure in order to examine the quality of the transformation result. Another technique is applied in order to ensure the quality of the transformation system is the reverse transformation algorithm .This chapter also proved the notion of completeness of our transformation system formally. Moreover this chapter showed some criteria that our transformation system satisfied flexibility, scalability and accuracy.

CHAPTER TEN: CONCLUSION AND FUTURE WORK

Objectives

- Present the thesis conclusion.
- Show the limitations of the approach.
- Highlight areas for future research.

10.1 Introduction

This chapter provides a brief summary of our work and concludes the thesis, addressing the limitations of the approach and discussing potential directions for further development.

10.2 Thesis summary

Database interoperability is the ultimate target that our research tried to solve. This considers allowing different databases to be integrated in a semantic method. To this aim, the thesis presented a framework explaining how databases can be transformed into ontologies. More specifically, it built a semi-automatic generic approach that provides a transformation system from a relational database to an ontology with minimal human effort. The significant contribution of this thesis has therefore been to demonstrate a novel stratified methodology with multi-database source mechanism that can significantly produce the ontology with the most database characteristics.

We have contributed the fundamental innovations in developing the solution to database integration are through the use of semantic web methodologies. Our proposed solution is to rely on information from the database relational model tables, primary and foreign key structure and the extended relationship model (cardinality restrictions, IS-A hierarchies). Theoretically our approach infers all, and only, the semantics implied by the table's schema based on the EER model.

Contrary to the legacy solutions for database integration such as database warehouses, federated databases focus more on physical mapping between databases, neglecting the semantic issues. However, in our approach we have analysed database schema in order to acquire explicit semantics that are necessary in constructing a transformation system, and we utilised ontology capabilities in improving semantic database integration.

10.2.1 Main Contributions

This thesis discusses many major contributions:

- Constructing a transformation system (relational model to ontological model).
- Creating a transformation system (conceptual database model to ontological model).
- Building a system to integrate both logical and conceptual database models to produce ontology.
- Implementing an automatic tool to transform SQL-DDL to OWL ontology.

Other secondary contributions:

- An intensive review of database models is provided, including definitions and structures.
- A comprehensive overview of ontological languages - their importance, structure and capabilities – is given.
- A comparative analysis of database models in order to show the limitations of each model.
- A comprehensive review of existing database transformation approaches. Several existing transformation approaches were analysed and compared in order to fully understand their strengths and weaknesses, and to create a new improved approach that is successful in both utilising the strengths of the approaches and overcoming their weaknesses. This analysis includes both theoretical work and existing practical transformation approaches.
- A comparison of database models with the ontological model, in order to discover the disparities in the concepts and aims of the two models in order to improve our transformation rules.
- Improvement of our transformation rules system through:

  - Formal methods eliminating the possibility of syntactic and semantic ambiguities.
  - The use of IF THEN forms in our rules, allowing the reader to easily comprehend the rules and enabling the designer to implement and modify the rules more easily.
  - The discovery of common mistakes in the analyses of relational design.
  - Completeness with respect to all possible relations and relationships.
  - Implementation of all components of our system based on a relational source.
  - Definition of our transformation rules in a way that ensures suitability between SQL-DDL from the database side, and OWL-DL from the or ontological side.

- Illustration of the application of our system to a case study.
- Achievement of high-quality, efficient results throughout the application of the system to real-world scenarios.
- Production of an OWL-DL ontology which promises decidability in the reasoning process.
- Substantiation with evidence of counter-examples that many approaches omit, mapping the SQL primitives to their corresponding OWL terms.
- As many approaches base their rules on enumerating examples their results may have produced incorrect rules, using special cases from a particular domain which cannot be generalised to comprise all possible cases. We therefore avoid the influence of domain-specific examples by building our rules on formal proof for all possible cases of both relations and relationships.
- Our rules are not domain-specific, and can therefore work with any relational schema.
- The capture of the semantics available in the relational model (SQL-DDL), and the capture of the missing semantics from the EER model are our rule contribution.
- The involvement of domain experts is required in capturing the semantics of the EER model that cannot be obtained from the relational model. Only two cases

of relationship identification need database designer participation; in the case of a well-designed database there is no need for domain expert intervention.

- Database integration is significant; to attain this objective we enrich our system with formal, scalable, functional and accurate criteria in order to construct a vital ontology.
- The consideration by our system of more complex constraints in a database in order to determine the inheritance and multiple-inheritance relationships among classes.

10.3 Evaluation of our approach

The main obstacle to evaluating our result was the absence of a benchmark, since there is no optimal model for any problem space, and there is no gold design for both database and ontology. Specification is the important factor that shapes database design, whereas ontology design tends to be more abstract; the best solution was therefore to create an ontology manually through a domain expert and use it as benchmark. Consequently we provided a domain expert with a relational database schema (SQL-DDL) and database specification or a conceptual mode of database, enabling him to generate an OWL ontology manually. A domain ontology developed two suggested benchmark ontologies; the first was an optimal solution considering high-level modelling; e.g. generalisation and multiple-inheritance, whereas the second suggested ontology was restricted to database specification. The two ontologies were subsequently developed by a domain expert and used as benchmarks for the evaluation process; the comparison then took place between the outcome of our system and the results of other approaches.

We proved that our transformation system is complete, as it takes into consideration all the possible interaction combinations between primary and foreign keys in our rules.

We additionally analysed the loss of semantics caused by the transformation, and used reverse transformation, which transforms the resulting ontology back to the relational database in order to see if the transformation is reversible or not. If reverse transformation yields the original relational database then we can consider the system to be accurate. Indeed, some of the semantics captured in a relational database will

necessarily be lost once the transformation is applied, however the percentage of lost semantics is trivial and does not require exploration. We therefore prove that our system is correct, as it is total and injective.

10.4 Criteria for achieving success

There are many criteria that prove our approach's success. Firstly, our approach is domain independent, therefore, any relational database can utilise our system to be published in Semantic Web. Secondly, our approach's rules produce a generic ontology design not restricted by a specific language. Thirdly, our approach is maintained to be scalable therefore the size of database would not affect our approach's performance. Fourthly, our approach contains a notion of completeness since it is built upon all the possible cases of the database model. Finally, the validation system of our approach will guarantee the correctness.

10.5 Limitations

The limitations of this thesis can be divided into two groups; the first based on the limitation of the sources' capability and the second being the general limitations of the approach.

The initial deficiencies of our approach stemmed from the disparities between database modelling and ontological modelling, as an ontology is semantically richer than a database. One of the disparities considered to be a major obstacle to our rules was the fact that databases operate in a 'closed world' scenario, whereas ontologies are usually based on an 'open world' assumption. Another crucial factor limiting the resulting ontology is the difference in the expressive power of the two languages used in SQL-DDL (database) and OWL (ontology) in terms of their capabilities [45]. In the evaluation chapter, we therefore observed the superiority of the manual ontology produced by the domain expert compared to the ontologies produced by automatic transformation systems. Another limitation may arise due to the original database design, meaning the database is built upon a pure relational model (ER) linked with the inheritance concept. Indeed this kind of database design will generate a database-

wearing ontological template. We therefore concede that the quality of ontologies built from the transformation system would not successfully match that of manual ones. However it takes too long and requires more effort to create an ontology from scratch, compared to the time needed to improve the ontology produced by our system.

The general limitations of the research are as follows:

- The focus of this study was predominantly on relational database applications, which are prone to schema modifications.
- The success of the transformation system is based on the amount of semantics obtained from database sources which in turn is associated with the age of the database and the experience of its developers.
- For the purposes of this study, the system is concerned with and restricted to SQL-DDL primitives owing to the vendor-based specifications of many SQL statements.

10.6 Future work

The following list highlights areas of research that are worth pursuing:

i. We would like to reduce human intervention in our transformation system to the minimal unavoidable cases; this can be achieved by identifying the cases that can be converted from semi-automatic to automatic without compromising the quality of the system's result.

ii. We also plan to extend the automatic transformation system to include de-normalised databases by capturing functional dependencies from database content.

iii. We plan to use Word.net in the enhancement process, which provides a rich and suitable nomenclature for classes and properties in order to utilise them in both string and linguistic matching within the ontological alignment methodology.

iv. We would like to modify the system to include a wrapper system, which is capable of rationalising the schema modification and propagating changes.

v. This research would involve working across non-database sources, e.g. HTML, XML etc.

vi. Making further research in ontology alignment and ontology mapping techniques in order to integrate them to our system.

There are interesting challenges with advanced SQL topics, in addition to SQL-DDL primitives, which are worth future investigation. These include:

i. Converting SQL queries from the database side to SPARQL from the ontology side.

ii. Using SWRL to represent database triggers and embedded procedures (PL/SQL).

REFERENCES

1- C. J. Date, "*An introduction to database systems*". Boston, MA; London, Pearson/Addison-Wesley, 2004.

2- R. N. Elmasri, and B. Shamkant, "*Fundamentals of database systems*", Boston, MA London, Addison Wesley, 2007.

3- H. Garcia-Molina, J. D. Ullman, and J.Widom, "*Database Systems"*, Upper Saddle River, N.J.: Pearson Prentice Hall, 2009.

4- A. K. Silberschatz, F.Henry, and S. Sudarshan, *"Database system concepts"*, New York, London, McGraw-Hill, 2006

5- W. Hu, and Y. Qu, "Discovering Simple Mappings Between Relational Database Schemas and Ontologies", Springer-Verlag Berlin Heidelberg, 2007,pp. 225–238.

6- S. Upadhyaya, and P.Kumar, "*ERONTO: A Tool for Extracting Ontologies from Extended E/R Diagrams*", SAC'05 ACM Symposium on Applied Computing, Santa Fe, New Mexico, USA, March 2005.

7- G. Antoniou, and F.V. Harmelen, "*Web Ontology Language: OWL*", Presented at Handbook on Ontologies, pp.67-92, 2004.

8- T. Berners-Lee, J. Hendler, and O. Lassila, "*The Semantic Web*", Scientific Am, May pp. 34–43, 2001.

9- D. Beckett, "*The Design and Implementation of the Redland RDF Application Framework*", In Proceedings of 10th International World Wide Web Conference, Hong Kong, May 2000.

10- T. Bray, J. Paoli, and C.M. Sperberg-McQueen, "*Extensible Markup Language (XML)*", Presented at World Wide Web Journal, pp.27-66, 1997.

11- D. Brickley, and R. Guha, *"Resource Description Framework (RDF) Schema specification*", 2000. http://www.w3.org/TR/RDF-schema.

12- O. Corcho, and A. Gómez-Pérez, "*Solving Integration Problems of E-Commerce Standards and Initiatives through Ontological Mappings*", In Proceedings of IJCAI 2001 Workshop on E-Business & the Intelligent Web, Seattle, USA, 2001.

13- M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "*OWL Web Ontology Language 1.0 Reference*", http://www.w3.org/TR/owl-ref/, 2002.

14- D. Dou, D. McDermott, and P. Qi, "*Ontology Translation on the Semantic Web*", Presented at on Data Semantics Journal, 3360:35–57, 2005.

15- D. Fensel, F.V. Harmelen, I. Horrocks, D.L. McGuinness, and P.F. Patel-Schneider, "*OIL: An Ontology Infrastructure for the Semantic Web*", Presented at IEEE Intelligent Systems, pp.38-45,2001.

16- D. Fensel, "*Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*", Springer, 2001.

17- F. van Harmelen, P.F. Patel-Schneider, and I. Horrocks (Editors), "*Reference Description of the DAML+OIL Ontology Markup Language*", http://www.daml.org/2000/12/reference.html, 2000.

18- O. Gotoh, "*An Improved Algorithm for Matching Biological Sequences*", Presented at Journal of Molecular Biology, 162:705-708, 1982.

19- O. Lassila, and R. Swick, "*Resource Description Framework (RDF) model and syntax specification*", 1999. http://www.w3.org/TR/REC-rdf-syntax.

20- M. Li, and M. Baker, "*The Grid: Core Technologies*", John Willey & Sons England , 2005.

21- A. Maedche, and S. Staab, "*Ontology Learning for the Semantic Web*", Presented at IEEE Intelligent Systems, pp.72-79, 2001.

22- B. McBride, S. Staab, and R. Studer (eds.), "*The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS, in: The Handbook on Ontologies in Information Systems*", Springer-Verlag, 2003.

23- D.L. McGuinness, and F. van Harmelen, "*OWL Web Ontology Language Overview*", Technical report, W3C, http://www.w3.org/TR/owl-features/, February 2004.

24- D.L. McGuinness, R. Fikes, L.A. Stein, and J.A. Hendler, "*DAML-ONT: An Ontology Language for the Semantic Web*", In Proceedings of Spinning the Semantic Web, pp.65-93, 2003.

25- G.A. Miller, ―*WordNet: A Lexical Database for English*", presented at Commun. ACM, pp.39-41, 1995.

26- H. Mihoubi, A. Simonet, and M. Simonet, "*An Ontology Driven Approach to Ontology Translation*", In Proceedings of DEXA, pp.573-582, 2000.

27- R. Neches, R. Fikes, T.W. Finin, T.R. Gruber, R.S. Patil, T.E. Senator, and W.R. Swartout, "*Enabling Technology for Knowledge Sharing*", Presented at AI Magazine, pp.36-56, 1991.

28- M. Paolucci, T. Kawamura, T.R. Payne, and K.P. Sycara, "*Semantic Matching of Web Services Capabilities*", In Proceedings of International Semantic Web Conference, pp. 333-347, 2002.

29- P. F. Patel-Schneider, P. Hayes, and I. Horrocks, "*OWL Web Ontology Language: Semantics and Abstract Syntax*", W3C Recommendation, February, 2004. (Available

via http://www.w3.org/TR/2003/WD-owl-semantics-20030331/ , last visited March 2009).

30- M. Smith, C.Welty, and D. McGuinness, "*OWL Web Ontology Language Guide*", http://www.w3.org/TR/2003/WD-owl-guide-20030331/.

31- D. Tidwell, "*Web Services-The Web's Next Revolution*", IBM Web Service Tutorial, 29 Nov. 2000, http://www-106.ibm.com/developerworks/edu/ws-dw-wsbasics-i.html.

32- M. Uschold and M. Gruninger, "*Ontologies: Principles, Methods and Applications*", Knowledge Engineering Review, vol. 11, no. 2, June 1996.

33- M. Vermeer, and P. Apers, "*Object-Oriented Views of Relational Databases Incorporating Behaviour*", Proceedings of the 4th International Conference on Database Systems for Advanced Applications (DASFAA), Singapore, April 11-13, pp. 26-35, 1995.

34- A. Behm, and A. Gepper,t K. Dittrich, "*On the Migration of Relational Schemas and Data to Object-Oriented Database Systems*", in Proceeding of the 5th Int. Conference on Re-Technologies for Information Systems Klagenfurt, pp. 13-33, December 1997.

35- V. Kashyap, "*Design and creation of ontologies for environmental information retrieval*", 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), Banff, Canada, October 1999.

36-Stojanovic, L., Stojanovic, N., and Volz, R. "*Migrating Dataintensive Web Sites into the Semantic Web*", Proc. 17th ACM Symposium on Applied Computing, Madrid, 2002.

37- G. Dogan, and R. Islamaj,"*Importing Relational Databases into the Semantic Web*", URL:http://www.mindswap.org/Webai/2002/fall/Importing_20Relational_20Databases_20into_20the_20Semantic_20web.html

38- D.L. Rubin, M. Hewett, D.E. Oliver, , T.E. Klein, and R.B. Altman, "*Automatic data acquisition into ontologies from pharmacogenetics relational data sources using declarative object definitions and XML*". In: Proceedings of the Pacific Symposium on Biology, Lihue, HI, 2002.

39- Astrova, I. "*Reverse Engineering of Relational Databases to Ontologies*" , Proc. 1st European Semantic Web Symposium (ESWS), Heraklion, Crete, Greece, LNCS, 327–341, 2004.

40- I. Astrova, and B. Stantic, "*An HTML Forms driven Approach to Reverse Engineering of Relational Databases to Ontologies"*, In: proceeding of the 23rd IASTED International Conference on Databases and Applications (DBA), eds. M. H. Hamza, Innsbruck, Austria, pp. 246- 251, 2005.

41- M.li, X.Du, and S.Wang "*Learning Ontology from Relational Databases*" Machine learning and Cybernetics, volume 6, pp3410-3415, 2005.

42- S. Benslimane, D. Benslimane, And M. Malki, "*Acquiring OWL Ontologies from Data-Intensive Web Sites*", USA.ACM, Palo Alto, California, July 11-14, 2006.

43- K. Sonia, and S. Khan, "*R2O Transformation System: Relation to Ontology Transformation for Scalable Data Integration*", in Proceedings of IDEAS08, Coimbra, Portugal, September 2008.

44- S. Upadhyaya and P.Kumar, "*ERONTO: A Tool for Extracting Ontologies from Extended E/R Diagrams*", in Proceedings of SAC'05 ACM Symposium on Applied Computing, Santa Fe, New Mexico, USA, March 2005.

45- S.H. Tirmizi, J. Sequeda and D. Miranker, "*Translating SQL Applications to the Semantic Web*" In S.S. Bhowmick, J. Küng, and R. Wagner (Eds.) DEXA 2008, LNCS 5181, Springer-Verlag, Berlin, pp. 450 – 464, 2008.

46- I. Astrova, , N. Korda, , and A. Kalja, "*Rule-Based Transformation of SQL Relational Databases to OWL Ontologies*", In: Proceedings of the 2nd International Conference on Metadata & Semantics Research, October 2007.

47- Z. Xu, X. Cao, Y. Dong, and W. Su, "*Formal Approach and Automated Tool for Translating ER Schemata into OWL Ontologies*", LNAI 3056, Springer-Verlag Berlin Heidelberg, pp. 464–476, 2004.

48- T. Gruber, "*Toward Principles for the Design of Ontologies Used for Knowledge Sharing*", In International Journal Human-Computer Studies 43, pp 907-928, March, 1993.

49- V. Sugumaran, and V. C. Storey, "*The Role of Domain Ontologies in Database Design: An Ontology Management and Conceptual Modelling Environment*", ACM Transactions on Database Systems, Vol. 31, No. 3, Pages 1064–1094, September 2006.

50- N. Noy, and A. Rector, "*Defining N-ary Relations on the Semantic Web*". W3C Working Group Note (13/12/2010), http://www.w3.org/TR/swbp-n-aryRelations/

51- M. Dean, and G. Schreiber, "*OWL Web Ontology Language Reference*". W3C Recommendation (13/12/2010), http://www.w3.org/TR/owl-ref/

52- V. Sugumaran, and V. C. Storey, "*Ontologies for Conceptual Modelling: their Creation, Use, and Management*", Data & Knowledge Engineering No.42, Pages 251–271, 2002.

53- J. F. Sequeda, S.H. Tirmizi, O. Corcho, and D.P. Miranker, "*Direct Mapping SQL Databases to the Semantic Web*". Technical Report 09-04, The University of Texas at Austin, Department of Computer Sciences.

54- M. Sabou, "*Extracting Ontologies from Software Documentation: A Semi-Automatic Method and Its Evaluation*" in Proc. Workshop on Ontology Learning and Population, Valencia, Spain, August 2004.

55- A. Buccella, M. R. Penabad, F. J. Rodríguez, A. Fariña, A. Cechich, "*From Relational Databases to OWL Ontologies*". Procs of the 6th Russian Conference on Digital Libraries. Pushchino, Rusia, 2004.

56- H.E. Ghalayini, M. Odeh, R. McClatchey, "*Engineering conceptual data models from domain ontologies: A Critical Evaluation*", in Proc. 4th International Conference on Computer Science and Information Technology, Amman, Jordan, April 2006.

57- Fonseca, F. and J. Martin, "*Learning the Differences Between Ontologies and Conceptual Schemas Through Ontology-Driven Information Systems*" in Proc. Journal of the Association for Information Systems - Special Issue on Ontologies in the Context of IS Volume 8, Issue 2, Article 3, pp. 129–142, February 2007.

58- B. Motik, I. Horrocks , and U. Sattler, "*Bridging the Gap between OWL and Relational Databases*", Proceedings Journal of Web Semantics Web Semantics: Science, Services and Agents on the World Wide Web, Volume 7, Issue 2, pp. 74-89, April 2009.

59- N. Guarino , "*Formal Ontology and Information Systems*", Proceedings of FOIS'98, Trento, Italy, pp. 3-15, June 1998.

60- C. Bizer ,and R. Cyganiak "*D2R Server – Publishing Relational Databases on the Semantic Web*", In Poster at the 5th International Semantic Web Conference, (2006) , (30/10/2010), http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/

61- J F. Sequeda, S. Tirmizi, and D. Miranker, "*SQL Databases are a Moving Target".* Position Paper for W3C Workshop on RDF Access to Relational Databases", Cambridge, MA, USA, 2007.

62- M. Taye, N. Alalwan, "Ontology Alignment Technique for Improving Semantic Integration", Proceedings in the Fourth International Conference on Advances in Semantic Processing, Florence, Italy, pp. 13-18, October 2010.

63- I. Astrova, N. Korda, and A. Kalja, "*Storing OWL Ontologies in SQL Relational Databases*", international journal of electrical, computer, and systems engineering, volume 1, number, pp. 242- 2474 ,2007.

64- H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner, "*Ontology-based integration of information - a survey of existing approaches*", In Workshop: Ontologies and Information Sharing, pp. 108-117, 2001.

65- N. Chatterjee, and M. Krishna, "*Semantic Integration of Heterogeneous Databases on the Web*" Proceedings of the International Conference on Computing: Theory and Applications, 2007.

66- R. Volz , S. Handschuh , Siegfried H , S. Staab , L. Stojanovic, and N. Stojanovi, "*Unveiling the Hidden Bride: Deep annotation for Mapping and Migrating Legacy Data to the Semantic Web*", Proceedings of Web Semantics: Science, Services and Agents on the World Wide Web, pp. 187–206, 2004.

67- Y. An, Alex Borgida, and J. Mylopoulos. "*Refining the Semantic Mappings from Relational Tables to Ontologies*". In Proceedings of 2nd International workshop on Semantic Web and Databases in conjunction with Very Large Databases, Toronto, Canada, LNCS 3372, Springer-Verlag, pages 84-90, 2004.

68- Y. An, A. Borgida, and J. Mylopoulos. "*Discovering and Maintaining Semantic Mappings between XML Schemas and Ontologies*", Proceedings of Journal of Computing Science and Engineering. Korean Institute of Information Scientists and Engineers, 2008.

69- D. Jurić, M. Banek, and Z. Skočir, "*Uncovering the Deep Web: Transferring Relational Database Content and Metadata to OWL Ontologies*", In Proceedings of Knowledge-Based Intelligent Information and Engineering Systems 12th International Conference, Zagreb, Croatia, September, pp. 456–463, 2008.

70 – O. Jautzy, "*Interoperable Databases: a Programming Language Approach*", in Proceedings from International Symposium on Database Engineering and Applications, Montreal, Que., Canada, pp. 63 – 71, August 1999.

71- S. Ram, and J. Park "*Semantic Conflict Resolution Ontology (SCROL): An Ontology for Detecting and Resolving Data and Schema-Level Semantic Conflicts*", IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 2, February 2004.

72- A.S. Aparicio, O.L.M. Farias, and N. Santos, "*Applying Ontologies in the Integration of Heterogeneous Relational Databases*". In Proc. Australasian Ontology Workshop, Sydney, Australia, vol. 58, pp.11-16, 2005.

73- M. Baglioni1, M. Vittoria Masserotti, C. Renso, and L. Spinsanti, "*Building Geospatial Ontologies from Geographical Databases*", LNCS 4853, Springer-Verlag Berlin Heidelberg, pp. 195–209, 2007.

74- J. Barrasa, O. Corcho, and A. Gómez-Pérez, "*Fund Finder: A Case Study of database-to-ontology mapping*", Proceedings Semantic Integration Workshop (ISWC), Sanibel Island, US, 2003.

75-D. Yeh, and Y. Li, "*Extracting Entity Relationship Diagram from a Table-based Legacy Database*", in Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR), 2005.

76- R. Alhajj, "*Extracting the Extended Entity-Relationship Model from a Legacy Relational Database*", Elsevier Science, Information Systems 28, pp 597–618, 2003.

77- Y. An, A. Borgida, and J. Mylopoulos. "*Inferring Complex Semantic Mappings between Relational Tables and Ontologies from Simple Correspondences*", In Proceedings of On The Move to Meaningful Internet Systems, Agia Napa, Cyprus, LNCS 3761, Springer Verlag, pp. 1152-1169, 2005.

78- j. Trinkunas, and O. Vasilecas. "*Building Ontologies from Relational Databases Using Reverse Engineering Methods*". In Proceedings of the International Conference on Computer Systems and Technologies (CompSysTech '07), Bulgaria, 2007, pp.1-6.

79- C. Bizer and F. U. Berlin, "*D2R MAP – A Database to RDF Mapping Language*", Budapest, Hungary, May, pp 20-24, 2003.

80- P.A. Champin, G.-J. Houben, and P. Thiran, "*Cross: An OWL Wrapper for Reasoning on Relational Databases*" LNCS 4801, pp. 502–517, 2007.

81- R. Ghawi and N. Cullot, "*Database-to-Ontology Mapping Generation for Semantic Interoperability*", VLDB '07, Vienna, Austria, September, 2007.

82- N. Cullot, R. Ghawi, and K. Yétongnon, "*DB2OWL : A Tool for Automatic Database-to-Ontology Mapping*", In SEBD, pp.491-494, 2007.

83- Z. Xu, S. Zhang, and Y. Dong, "*Mapping between Relational Database Schema and OWL Ontology for Deep Annotation*", Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2006.

84- C. Hu, H. Li, X. Zhang, and C. Zhao, "*Research and Implementation of Domain-Specific Ontology Building from Relational Database*", The Third China Grid Annual Conference, pp 289-293, 2008.

85- G. Shen, Z. Huang, X. Zhu, and X. Zhao, "*Research on the Rules of Mapping from Relational Model to OWL*", Proceedings of the Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, November, 2006.

86 – I. Astrova, "*Towards the Semantic Web – An Approach to Reverse Engineering of Relational Databases to Ontologies*", proceedings of the 9th East-European Conference on Advances in Databases and Information Systems (ADBIS), Tallin, September, 2005.

87- D. Jurić, M. Banek, and Z. Skočir, "*Uncovering the Deep Web: Transferring Relational Database Content and Metadata to OWL Ontologies*", LNAI 5177, Part I, pp. 456–463, 2008.

88- C. Nyulas , and S. Tu, "*DataMaster – a Plug-in for Importing Schemas and Data from Relational Databases into Protégé*", In Proceedings of 10th International Protégé Conference.

89- M. Korotkiy , and J. L. Top, "*From Relational Data to RDFS Models*", International Conference on Web Engineering, LNCS, Vol. 3140, 2004.

90- C. Perez, d. Laborda and S. Conrad, "*Relational.OWL - A Data and Schema Representation Format Based on OWL*", the Second Asia- Pacific Conference on Conceptual Modelling (APCCM), New-castle, Australia, 2005.

91- E. Vysniauskas, and L. Nemuraite, "*Transforming Ontology Representation from Owl To Relational Database*", Information Technology and Control, Vol.35, No.3A, pp. 333 – 343,2006.

92- A. Ranganathan, and Z. Liu, "*Information Retrieval from Relational Databases using Semantic Queries*", CIKM'06, Arlington, Virginia, USA, pp. 820-82, November, 2006.

93- F.Cerbah, "*Learning Highly Structured Semantic Repositories from Relational Databases: the RDBTOONTO Tool*", Proceedings of the 5th European Semantic Web Conference (ESWC) on The Semantic Web: research and applications, 2008.

94- B. Habegger, "*Mapping a database into an ontology: an interactive relational learning approach*", IEEE 23rd International Conference on Data Engineering, pp. 1443-1447, 2007.

95- H. Zhuge, Y. Xing And P. Shi, "*Resource Space Model, OWL and Database: Mapping and Integration*" ACM Transactions on Internet Technology, Vol. 8, No. 4, Article 20, September 2008.

96 M. Krishna, "*Retaining Semantics in Relational Databases by Mapping them to RDF*", Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2006.

97- S. Das, E. I. Chong, G. Eadon, and J. Srinivasan, "*Supporting Ontology-based Semantic Matching in RDBMS*", Proceedings of the 30th VLDB, Conference, Toronto, Canada, pp. 1054- 1065, 2004.

98- J. Xu, and W. Li, "*Using Relational Database to Build OWL Ontology from XML Data Sources*", Proceedings in International Conference on Computational Intelligence and Security Workshops, pp 124-127, 2007.

99- N. Konstantinou, D. Spanos, M. Chalas, E. Solidakis and N. Mitrou, "*VisAVis: An Approach to an Intermediate Layer between Ontologies and Relational Database Contents*", Proceedings in International Workshop on Web Information Systems Modelling (WISM), 2006.

100- Wei Hu, and Yuzhong Qu "*Discovering Simple Mappings Between Relational Database Schemas and Ontologies*", In Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference ISWC/ASWC2007, Busan, South Korea, pp. 225-238, November, 2007.

101- A. C. Muñoz, and J. L. Aguilar "*Architecture for an Intelligent Distributed Database*", Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2005.

102- Olivier Jautzy, "*Interoperable Databases : a Programming Language Approach*", Proceedings from IDEAS '99 International Symposium on Database Engineering and Applications,1999.

103- Goksel Aslan, and Dennis McLeod "*Semantic Heterogeneity Resolution in Federated Databases by Metadata Implantation and Stepwise Evolution*", The VLDB Journa,l vol. .8, pp 120–132, 1999.

104- F. M. Al-Wasil W. A. Gray, and N. J. Fiddian, "*Establishing an XML Metadata Knowledge Base to Assist Integration of Structured and Semi-structured Databases*", the Seventeenth Australasian Database Conference (ADC), Hobart, Australia ,Vol. 49, 2006.

105- H. Müller J. Freytag, U. Leser, "*Describing Differences between Databases*" Proceedings Conference on Information and Knowledge Management (CIKM), Arlington, Virginia, USA, November 5–11, 2006.

106- M. Taye, "*Ontology Alignment Mechanisms for Improving Web-based Searching*", PhD thesis, De Montfort University, Leicester, UK, 2009.

107- N. Noy and D. McGuinness, "Ontology Development 101: A guide to creating your first ontology", Technical Report, Stanford University, Stanford, CA, US.

Appendix A

Normalisation:

The design of relational schema requires a way for evaluating its quality. Therefore, the normalisation can be a formal measure which show why one grouping of attributes into a relation schema may be better than other. This measure demonstrates the appropriateness or the goodness of database design.

Why we need normalisation?

- To minimise the redundancy data.
- To minimise anomalies update (insertion, deletion, and modification).

Before explaining the different level of normalisation we need to define some related concepts.

- **Functional Dependencies** (FD): for a given relation R, attribute set Y of R is functionally dependent on attribute set X of R if and only if each x-value in R has associated with it exactly y-value in R. in formally whenever two tuples agree on their X value, they also agree on their Y value.
    - Denoted by, $X \rightarrow Y$.
- **Normalisation of relations**: a process of analysing a given relation schemas based on their FDs and primary keys to satisfies a certain condition.
    - **Prime attribute**: given a relation R, an attribute A is a prime if A is contained in some key of R.
    - **transitive dependency**: Given a relation R, an attribute A of R is said to be transitively dependent on attribute set X of R if there exists an attribute set Y of R such that
        - $X \rightarrow Y$,
        - $Y \rightarrow A$,
        - $Y \nrightarrow X$ And $A \notin X \cup Y$.

- **Normalisation Forms:**
    - First normal form:

A relation is said to be in First Normal Form (1NF) if the domain of all its attributes are atomic.

- Second normal form:

The relation R is in Second Normal Form (2NF) if it is in 1NF and every non-prime attribute is fully functionally dependent on every key of R.

- Third normal form:

A relation is said to be in third normal form (3NF) if it is in 1NF and no non-prime attribute in R is transitively dependent on any key of R.

- The other normal forms:

There are other types of normal forms such as BOYCE-CODD Normal (BCNF), Forth Normal Form (4NF), and Fifth Normal from (5NF).

However, Database designers today pay attention to normalise their database to the 3NF. The reason for that is 1NF and 2NF database suffer from performance problem. Also the higher forms are hard to understand or difficult to detect.

Appendix B

The university SQL-DDL

Department table:
Create table department(dept_id int, dept_name varchar(70) not null,dept_phone varchar(20), primary key(dept_id));
Staff table:
Create table staff(staff_id int, staff_ family_name varchar(50) not null,dept_id int, manger_id int, n_id int not null unique, primary key(staff_id), foreign key (dept_id) references dept(dept_id) foreign key (manger_id references staff(staff_id)); foreign key (staff_id) references staff-details (staff_id));
Staff-details table:
Create table staff-details(staff_id int,staff_first_name varchar(10), staff_mid_name varchar(15), DOB date,address varchar(50),email varchar(50),ext phone varchar(10),homephone varchar(20) primary key(staff_id), foreign key (staff_id) references staff(staff_id));
Academic-staff table:
Create table academic-staff(n-id int unique not null, staff_ Post-held varchar check (Post-held in (‘ Teacher Assistance’, ‘Instructor’, ‘Assistant professor’, ‘Associate professor’, ‘Professor’) specialty varchar (40), primary key(staff_id), foreign key (staff_id) references staff(staff_id));
Student table:
Create table student(st_id int, st_name varchar(70) not null, sex char(1), module-name varchar(70) unique ,dept_id int not null, n-id unique not null, primary key(st_id),

foreign key (dept_id) references department (dept_id));

Graduate-student table:

Create table graduate-student(st_id int, research_area varchar(70),
primary key(st_id),
foreign key (st_id) references student(st_id)

Post-Graduate

Create table post- graduate (st_id int, project_ group varchar(70), staff-id unique not null
primary key(st_id),
foreign key (st_id) references student(st_id)
foreign key (staff-id) references academic-staff);

Hobby table:

Create table hobby(st_id int, hobby_name varchar(70),
primary key(st_id, hobby_name),
foreign key (st_id) references student(st_id));

Dependent table:

Create table dependent(d_id int, dependent_name varchar (20) not null, relationship varchar (20) DEFAULT 'Parent',
primary key(d_id, staff_id),
foreign key (staff_id) references staff(staff_id));

Course table:

Create table course(c_id int, course_name varchar(70) not null, dept_id int,
course-credit-hour int, co-offer int,
primary key(c_id)
foreign key (dept_id) references department (dept_id)
foreign key (co-offer) references course (c_id));

Registered table:

Create table registered(st_id int, c_id int,
primary key(st_id,c_id),
foreign key (st_id) references student(st_id),
foreign key (c_id) references course(c_id));

Section table:

```
Create table Section (st_id int, c_id int, staff_id int,
        primary key(st_id,c_id,staff_id),
        foreign key (st_id) references  student (st_id),
        foreign key (c_id) references course(c_id),
        foreign key (staff_id)
        references academic-staff (staff_id));
```

Appendix C

```
<owl:Class rdf:ID="Academic_staff">
    <rdfs:subClassOf rdf:resource="#staff"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:ID="annotationProperty_1">
    <rdf:type rdf:resource="&owl;AnnotationProperty"/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:ID="belongTo">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#department"/>
    <owl:inverseOf rdf:resource="#invBelongTO"/>
  </owl:ObjectProperty>
  <owl:Class rdf:ID="course"/>
  <owl:ObjectProperty rdf:ID="course_available">
    <rdfs:domain rdf:resource="#course"/>
    <rdfs:range rdf:resource="#Student"/>
    <owl:inverseOf rdf:resource="#student_registered"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="course_id">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#course"/>
    <rdfs:range rdf:resource="&xsd;int"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="course_name">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#course"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:ID="course_taught">
    <rdfs:domain rdf:resource="#course"/>
    <rdfs:range rdf:resource="#Teach"/>
```

```
    <owl:inverseOf rdf:resource="#has_course_teach"/>
  </owl:ObjectProperty>
  <owl:Class rdf:ID="department">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#invBelongTO"/>
        <owl:minCardinality rdf:datatype="&xsd;int">0</owl:minCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Dependent"/>
  <owl:Class rdf:ID="Graduate_Student">
    <rdfs:subClassOf rdf:resource="#Student"/>
  </owl:Class>
  <Graduate_Student rdf:ID="Graduate_Student_18"/>
  <owl:ObjectProperty rdf:ID="has_course_teach">
    <rdfs:domain rdf:resource="#Teach"/>
    <rdfs:range rdf:resource="#course"/>
    <owl:inverseOf rdf:resource="#course_taught"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has_staff_teach">
    <rdfs:domain rdf:resource="#staff"/>
    <rdfs:range rdf:resource="#Teach"/>
    <owl:inverseOf rdf:resource="#staff_is_teaching"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has_student_teach">
    <rdfs:domain rdf:resource="#Teach"/>
    <rdfs:range rdf:resource="#Student"/>
    <owl:inverseOf rdf:resource="#taught_by"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has_subordinate">
```

```
    <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
    <rdfs:domain rdf:resource="#staff"/>
    <rdfs:range rdf:resource="#staff"/>
    <owl:inverseOf rdf:resource="#has_superior"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="has_superior">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#staff"/>
    <rdfs:range rdf:resource="#staff"/>
    <owl:inverseOf rdf:resource="#has_subordinate"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="invBelongTO">
    <rdfs:domain rdf:resource="#department"/>
    <rdfs:range rdf:resource="#Student"/>
    <owl:inverseOf rdf:resource="#belongTo"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Post_Graduate">
    <rdfs:subClassOf rdf:resource="#Student"/>
    <rdfs:subClassOf rdf:resource="#Academic_staff"/>
</owl:Class>
<Post_Graduate rdf:ID="Post_Graduate_19"/>
<owl:DatatypeProperty rdf:ID="spacilty">
    <rdfs:domain rdf:resource="#Academic_staff"/>
</owl:DatatypeProperty>
<owl:Class rdf:ID="staff">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#staff_is_teaching"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

```
<owl:DatatypeProperty rdf:ID="Staff_id">
   <rdf:type rdf:resource="&owl;FunctionalProperty"/>
   <rdfs:domain rdf:resource="#staff"/>
   <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="staff_is_teaching">
   <rdfs:domain rdf:resource="#Teach"/>
   <rdfs:range rdf:resource="#staff"/>
   <owl:inverseOf rdf:resource="#has_staff_teach"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Student">
   <owl:equivalentClass>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#has_student_teach"/>
         <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
      </owl:Restriction>
   </owl:equivalentClass>
   <rdfs:subClassOf rdf:resource="&owl;Thing"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#belongTo"/>
         <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
<owl:DatatypeProperty rdf:ID="Student-id">
   <rdf:type rdf:resource="&owl;FunctionalProperty"/>
   <rdfs:domain rdf:resource="#Student"/>
   <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Student_name">
   <rdf:type rdf:resource="&owl;FunctionalProperty"/>
```

```
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:ID="student_registered">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#course"/>
    <owl:inverseOf rdf:resource="#course_available"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="taught_by">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#Teach"/>
    <owl:inverseOf rdf:resource="#has_student_teach"/>
  </owl:ObjectProperty>
  <owl:Class rdf:ID="Teach">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:onProperty rdf:resource="#has_course_teach"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#has_student_teach"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty rdf:resource="#staff_is_teaching"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
```

```
    </owl:Class>
</rdf:RDF>
```

Appendix D

Table for object properties score of University ontology

object	domain	range	I. Astrova, et al.	M. LI ,et al.	S.H. Tirmizi, et al.	SQL2OWL	Domain SQL specific	Domain enhanced
Has Name	person	Name	(-1,-0.25)	(-1,-0.25)	(-1,-0.25)	(-1,-0.25)		(1,0.25)
Has Address	person	Address	(-1,-0.25)	(-1,-0.25)	(-1,-0.25)	(-1,-0.25)		(1,0.25)
Co-offer	course	course	(1,0.5)	(1,-0.5)	(1,-0.5)	(1,-0.5)	1,.5	1,.5
manger	staff	staff	(1,-.05)	(1)	(1,0.5)	(1,0.5)	(1,0.5)	(1,0.5)
Has student transcript	transcript	student	(1,-0.25)	(1,-0.5)	(1,-0.25)	(1,0.25)	(1,0.25)	(1,0.25)
Has course transcript	transcript	course	(1,-0.25)	(1,-0.5)	(1,-0.25)	(1,0.25)	(1,0.25)	(1,0.25)
Has student section	section	student	(1,-0.25)	(1,0.25)	(1,-0.25)	(1,0.25)	(1,0.25)	(1,0.25)
Has ac-Staff section	section	ac-Staff	(1,-0.25)	(1,0.25)	(1,-0.25)	(1,0.25)	(1,0.25)	(1,0.25)
Has course section	section	course	(1,-0.25)	(1,0.25)	(1,-0.25)	(1,0.25)	(1,0.25)	(1,0.25)
Has staff	department	staff	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)
Has course	department	course	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)
Student belong dept	Student	department	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)	(1,0.25)
Student reg	Student	course	1	1	1	1	1	1
Course reg	Course	Student	1	1	1	1	1	1
Superfluous Relationship Has hobby	student	hobby	-0.5	-.05	-.05	--	--	--
total			11 – 8.5	11.5 – 9	11- 8.5	14-11.50	14.5-12	17.5
DO Sp			11/14.5 75%	11.5/14.5 79%	11/14.5 75%	14/14.5 96%		
DO EN			8.5/17.5 48%	9/17.5 51%	8.5/17.5 48%	11.5/17.5 65%		

Table for datatype properties score of University ontology

Datatype	domain	range	Datatype characteristic	Pk	Not null	unique	Not null& unique	I.Astrova, et al.	M. LI ,et al.	S.H. Tirmizi, et al.	SQL2OWL
Dept_id	department	int	Card=1	☑				.5+.25-.25	.5+.25	.5-.25	.5+.25
Dept Name	department	string	Card=1		☑			.5 +.25	.5+.25	.5+.25	.5+.25
Dept Phone	department	string	Functional					.5+.25	.5-.25	.5+.25	.5+.25
n-id	person	int	Card=1				☑	.5+.25-.25-.25	.5+.25 -.25	.5-.25-.25	.5+.25-.25
Family name	person	string	Card=1		☑			.5+.25-.25	.5+.25 -.25	.5+.25-.25	.5+.25-.25
Mid-name	person	string	Functional					.5-.25+.25	.5-.25	.5-.25	.5-.25+.25
First Name	person	string	Card=1		☑			.5+.25-.25	.5+.25 -.25	.5+.25-.25	.5+.25-.25
Staff-id	staff	int	Card=1	☑				.5+.25-.25	.5+.25	.5-.25	.5+.25
DOB	staff	date	Functional					.5+.25	.5-.25	.5+.25	.5+.25
Email	staff	string	Functional					.5+.25	.5-.25	.5+.25	.5+.25
Ext phone	staff	string	Functional					.5+.25	.5-.25	.5+.25	.5+.25
Home phone	staff	string	Functional					.5+.25	.5-.25	.5+.25	.5+.25
Post held	Academic-	strin	One of					.5+.25	.5	.5+.25	.5+.25

	staff	g									
Specialty	Academic-staff	string	Functional					.5+.25	.5-.25	.5+.25	.5+.25
Student id	Student	int	Card=1	☑				.5+.25-.25	.5+.25	.5-.25	.5+.25
sex	Student	string	Functional					.5+.25	.5-.25	.5+.25	.5+.25
Module name	Student	string	maxCard=1			☑		.5-.25	.5+.25	.5-.25	.5+.25
hobby	Student	string	===					.5-.25-.25	.5-.25	.5-.25-.25	.5+.25
Research area	Graduate-Student	string	Functional					.5+.25	.5-.25	.5+.25	.5+.25
Project group	Post-Graduate	string	Functional					.5+.25	.5-.25	.5+.25	.5+.25
Dependent id	Dependent	int	Card=1	☑				.5+.25-.25	.5+.25	.5-.25	.5+.25
Dependent name	Dependent	string	Card=1		☑			.5+.25	.5+.25	.5+.25	.5+.25
Course id	Course	int	Card=1	☑				.5+.25-.25	.5+.25	.5-.25	.5+.25
Course name	Course	string	Card=1		☑			.5+.25	.5+.25	.5+.25	.5+.25
Course credit hour	Course	int	Functional					.5+.25	.5-.25	.5+.25	.5+.25
grade	transcript	int	Functional					.5+.25	-.25	.5+.25	.5+.25
TOTAL								16.75-15.75	12.5-11.5	15-14	19.5-18.5
DO Sp							/19.5	85%	64%	76%	100%
DO EN							/19.5	80%	58%	71%	94%

Precision SDO

Approach	**I.Astrova, et al.**	**M. LI ,et al.**	**S.H. Tirmizi, et al.**	**SQL2OWL**
Class &subclass relationship	13/16	12/15	13/16	15/15
Object	13/14	13/14	13/14	13/13
Object domain	12/14	8/14	12/14	13/13
Object characteristic	10/18	12/13	7/11	13/13
Datatype	26/26	25/25	26/26	26/26
Datatype domain	25/26	24/26	25/26	26/26
Datatype characteristic	25/31	12/13	18/25	25/25
Precision SDO total	124/145	106/120	114/132	131/131
Precision SDO	85%	88%	86%	100%

Recall SDO

approach	I.Astrova, et al.	M. LI ,et al.	S.H. Tirmizi, et al.	SQL2OWL
Class &subclass relationship	13/15	12/15	13/15	15/15
Object	13/13	13/13	13/13	13/13
Object domain	13/13	8/13	13/13	13/13
Object characteristic	10/16	12/16	7/16	14/16
Datatype	26/26	25/26	26/26	26/26
Datatype domain	25/26	24/26	25/26	26/26
Datatype characteristic	25/25	12/25	18/25	25/25
Recall SDO total	125/134	106/134	115/134	132/134
Recall SDO	93%	97%	85%	98%

Precision EDO

Precision	I.Astrova, et al.	M. LI ,et al.	S.H. Tirmizi, et al.	SQL2OWL
Class	13/16	12/16	13/16	14/14
Object	12/15	12/15	12/15	12/14
Object domain	12/15	8/15	12/15	12/14
Object characteristic	14/22	6/14	7/20	15/15
Datatype	25/25	26/26	26/26	26/26
Datatype domain	21/26	20/26	21/26	22/26
Datatype characteristic	12/20	25/25	18/25	25/25
	109/139	109/137	109/143	126/134
Average	78%	79%	76%	94%

Recall EDO

Recall	Astrova, et al.	LI ,et al.	Tirmizi, et al.	SQL2OWL
Class	13/19	12/19	13/19	14/19
Object	12/14	12/14	12/14	12/14
Object domain	12/14	8/14	12/14	12/14
Object characteristic	9/15	9/15	7/15	11/15
Datatype	26/26	25/26	26/26	26/26
Datatype domain	21/26	20/26	21/26	22/26
Datatype characteristic	24/25	24/25	18/25	25/25
	117/139	110/139	113/139	122/139
average	84%	79%	81%	87%